uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática

2017-2018

Trabajo Fin de Grado

Eficiencia de almacenamiento basado en documentos vs almacenamiento relacional

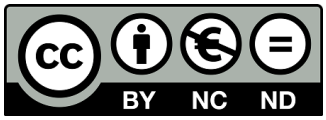
---

Miguel De Arriba Moreno

Tutor/es

Ana María Iglesias Maqueda

Leganés - 2018



RESUMEN

En este trabajo se realiza una comparación de rendimiento entre un sistema basado en documentos contra un sistema relacional. Para ello se realiza un caso práctico utilizando un conjunto de datos estructurados encontrado en Internet y se utilizan las tecnologías MongoDB y Oracle para llevar a cabo la comparativa. La realización del caso práctico cubre desde el análisis de los datos disponibles hasta el estudio de los resultados obtenidos tras ejecutar las consultas en ambos sistemas, pasando por el diseño de las estructuras necesarias para realizar los diseños de las bases de datos. Por último, a partir de los resultados obtenidos, se hace un estudio de las diferentes razones que pueden llevar a esos resultados.

Palabras clave: NoSQL, MongoDB, almacenamiento relacional, almacenamiento basado en documentos, Oracle, comparación, rendimiento, eficiencia.

DEDICATORIA

En primer lugar dar las gracias a mi familia, especialmente a mis padres y a mi hermana por todo el esfuerzo y confianza depositados en mí, así como por haberme enseñado que con esfuerzo y trabajo se puede lograr lo que uno se proponga.

También dar las gracias a los que ya no están, gracias por todo lo que me enseñaron, ellos también formaron una parte importante del camino recorrido y sin ellos habría sido incluso más complicado llegar hasta aquí.

A mis amigos de toda la vida Diego, Iván y Álex por haber estado ahí siempre, por haberme apoyado y distraído.

A mis compañeros de clase Fermín, Javi, Rubén, Christian, Dani y Quique. Gracias por todos esos momentos que hemos pasado juntos en la universidad o fuera de ella. Finalmente sobrevivimos a la 70j06 y nos graduamos ;)

Por supuesto también dar las gracias a Ainara, que aunque ella también fue compañera de clase, con el tiempo se convirtió en mucho más que eso. Gracias por haberme ayudado y aguantado todo este tiempo y por tu infinita paciencia conmigo, con lo pesado que me ponía diciendo que no nos iba a dar tiempo a entregar nada... Sin duda te deberían dar una matrícula de honor.

Finalmente, dar las gracias a Ana María Iglesias por haberme guiado y ayudado durante la realización de este trabajo y darme los consejos necesarios para que saliera mejor. Todo ello a pesar de que las condiciones quizás tampoco fueron las mejores...

Y, por último, si me olvido de alguien o alguien cree que debería haber sido añadido que me perdone, seguramente que si me conoce sabrá que soy un desastre... Pero en definitiva, GRACIAS A TODOS.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Planteamiento del problema y Motivación

En los últimos años, debido a la aparición de nuevas tecnologías como Big Data o Cloud Computing, las necesidades de las empresas han cambiado, ya que ahora buscan poder almacenar y tratar grandes volúmenes de información de manera eficiente. Por otro lado, también han aparecido distintos tipos de aplicaciones con unas necesidades muy concretas como alta concurrencia en las operaciones de lectura y escritura con una baja latencia o una alta capacidad de escalabilidad y disponibilidad. Todo ello supone un gran reto para las bases de datos relacionales, ya que, en un sistema de bases de datos relacional, las más utilizadas actualmente, el principal problema a la hora de querer escalar el sistema puede ser las relaciones que se desean mantener entre sus datos, almacenados lógicamente en diferentes estructuras.

Para intentar cubrir estas nuevas necesidades, han aparecido una gran variedad de tipos de bases de datos. Estas nuevas bases de datos, en general, tienen unas características muy diferentes a las bases de datos relacionales tradicionales, y han recibido el nombre de almacenes de datos NoSQL (una abreviación de Not Only SQL). Algunos aspectos diferentes de este nuevo tipo de bases de datos son la capacidad para realizar operaciones de lectura y escritura muy rápido, capacidad para almacenar grandes volúmenes de datos, fácilmente escalables y, generalmente, su coste suele ser muy inferior al de un sistema gestor de bases de datos relacional.

Todo ello ha llevado a que en los últimos años los almacenes de datos NoSQL hayan adquirido una gran popularidad, debido en parte a su facilidad de uso y a las nuevas bondades respecto a los sistemas gestores tradicionales. El problema surge a la hora de cuando una empresa necesita implantar una nueva base de datos y se plantea qué tipo va a utilizar, si un sistema SQL tradicional o un sistema NoSQL. Generalmente, esta cuestión no tiene una respuesta sencilla y requiere de un minucioso análisis de las necesidades que tiene la empresa y de los datos que se quieren almacenar. Sin embargo, muchas empresas optan por una de las opciones porque "está de moda" o por han oído que una empresa de la competencia lo ha hecho así sin detenerse a pensar cuáles son sus necesidades, incurriendo en muchos casos en una mala elección que puede tener un gran impacto en la marcha negocio.

En este trabajo de Fin de Grado se intenta plasmar las diferencias entre los distintos tipos de bases de datos existentes en el mercado, tanto los sistemas tradicionales como los nuevos sistemas NoSQL, cuáles son los puntos buenos y malos de cada uno de los tipos y llevar a cabo una comparativa entre dos sistemas aparentemente opuestos. Para ello, se va a realizar un caso práctico de estudio de eficiencia en consultas complejas implementadas mediante una base de datos relacional y un sistema de almacenamiento NoSQL documen-

tal. Todo ello para poner de manifiesto que en muchas ocasiones no es tan sencillo como elegir un sistema u otro aleatoriamente, sino que esa elección requiere un minucioso análisis de las características y necesidades de cada negocio. Esta comparativa será un factor más a valorar a la hora de elegir entre ambas tipologías de bases de datos para el dominio concreto que se estudia y dominios de características similares.

1.2. Objetivos

Los principales objetivos del presente proyecto se presentan a continuación:

- Realizar un estudio de las diferentes tipologías de bases de datos existentes en el mercado, indicando las principales características de cada sistema.
- Elegir dos tipos de almacenamiento de bases de datos bien diferenciados para realizar un estudio de rendimiento de consultas complejas.
- Selección de caso práctico a implementar y consultas complejas sobre los datos.
- Diseño e implementación de caso práctico y consultas seleccionadas en los sistemas de almacenamiento seleccionados previamente.
- Estudio del rendimiento de las consultas implementadas en ambos sistemas.

1.3. Estructura del documento

El presente documento está dividido en cinco secciones, las cuales se resumen a continuación:

- Capítulo1 “Introducción”: se realiza una introducción del documento y se establecen cuáles son los objetivos que se pretenden alcanzar con la realización del proyecto.
- Capítulo 2 “Gestión del Proyecto”: se establece la planificación temporal, detallando las diferentes etapas del proyecto, así como un análisis costes del mismo.
- Capítulo 3: “Estado del Arte”: en este capítulo se presentan los distintos sistemas gestores de bases de datos que existen actualmente, tanto tradicionales como NoSQL, así como un análisis del entorno socio-económico y del marco regulador.
- Capítulo 4 “Análisis”: en este capítulo se explica todo el proceso de análisis que se ha llevado a cabo para la realización del proyecto; desde la ilicitación de requisitos hasta el análisis del entorno operacional previo a la elección de los sistemas de bases de datos que se iban a utilizar.
- Capítulo 5: “Diseño”: en este capítulo se detalla el proceso de diseño que se ha realizado durante el proyecto tanto de la base de datos NoSQL como de la base de datos relacional. Asimismo, se detallan las consultas que se van a realizar en la comparativa y se proporciona una breve explicación de las mismas.
- Capítulo 6: “Implementación”: en este capítulo se detalla la implementación de las consultas tanto en MongoDB como en Oracle.
- Capítulo 7: “Evaluación”: en este capítulo se explica el diseño de la experimentación que se ha llevado a cabo, así como la explicación de los resultados obtenidos en la ejecución de las consultas.
- Capítulo 8 “Conclusiones y Trabajos Futuros”: se explican las conclusiones que se han obtenido a partir del trabajo realizado. De igual manera, se establecen posibles líneas que se podrían utilizar como continuación de este proyecto en el futuro.
- Capítulo 9: “Work Summary”: en este capítulo se presenta un resumen de todo el proyecto realizado en inglés.

Por último, se incluye un anexo con los acrónimos y términos utilizados en el documento; un anexo adicional con el proceso de instalación que se ha realizado tanto de Oracle como de MongoDB y ,finalmente, los diagramas de GANTT que se han hecho en la planificación.

2. GESTIÓN DEL PROYECTO

En este punto se explica detalladamente el proceso de gestión que se ha seguido durante la realización del proyecto. En primer lugar se mencionará cómo se ha realizado la gestión del software, es decir, la metodología que se ha seguido para la generación del mismo y el ciclo de vida del proyecto organizándolo en fases. Posteriormente, se realiza la planificación inicial de proyecto para finalizar con la planificación final real que se ha llevado a cabo y el cálculo de los costes relativos al proyecto.

2.1. Gestión Software: Metodología y Ciclo de Vida

Con el fin de evitar inconvenientes durante el desarrollo del software así como para evitar errores en el software realizado, conviene elegir una buena metodología previamente a comenzar un proyecto software.

Una metodología es el proceso que se sigue para diseñar un programa de software teniendo en cuenta diversos factores como pueden ser los costes, la planificación o la calidad del producto. Elegir una buena metodología al comienzo de un proyecto es importante, sin embargo, esta elección no asegura el éxito del mismo, ya que este dependerá en gran medida de las decisiones de los responsables de dicho proyecto. No obstante, realizar una buena elección sí que puede influir positivamente durante el desarrollo de un proyecto.

Aunque dada la gran variedad de metodologías existentes y sus distintas variaciones esta elección puede no ser siempre sencilla, se deben conocer las necesidades de cada proyecto particular para, a partir de ellas, elegir la metodología que más se adecúe a las mismas.

Así, dadas las características del presente proyecto, se decidió elegir una metodología que siguiera el modelo en cascada, en la cual las diferentes etapas del proyecto quedan definidas claramente:

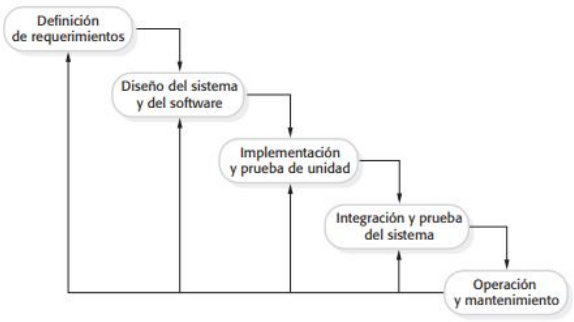


Fig. 2.1. Ciclo de vida en cascada [1]

Como se puede apreciar en la imagen, este ciclo de vida del software consta de cinco

fases separadas y consecutivas: Análisis, Diseño, Implementación, Pruebas y Mantenimiento; aunque la fase de mantenimiento no se tendrá en cuenta para el proyecto. Como norma general, la siguiente fase no debe comenzar sino se se ha terminado la fase anterior, no obstante, en la práctica las etapas se solapan y comparten información [1] que permite hacer cambios sobre la etapa anterior. Cabe destacar que, ocasionalmente, el descubrimiento de errores puede llevar a repetir etapas anteriores completamente.

Aunque esta metodología es un clásico que se lleva utilizando muchos años en el desarrollo de software, se ha optado por ella anteponiéndola a otras que en la actualidad se usan más, como por ejemplo las metodologías ágiles, ya que estas últimas están más centradas en la realización de pequeños sprints y en la involucración del cliente. Sin embargo, estos aspectos carecían de importancia para nuestro proyecto.

2.2. Planificación Inicial

Como ya se ha especificado, el proyecto se divide en cinco fases bien diferenciadas. Estas son las fases de análisis, diseño, implementación, evaluación y documentación. Adicionalmente se ha añadido una fase de documentación que servirá para realizar los documentos relativos a todo el proyecto. Dada esta jerarquización de las tareas que se deben realizar para llevar a cabo el proyecto, se debe establecer una planificación temporal que será la que se siga a lo largo de la realización del mismo.

En primer lugar, mencionar que este trabajo se está realizando con motivo de la finalización de los estudios que el alumno cursando: "Grado en Ingeniería Informática" impartido por la Universidad Carlos III de Madrid. Es por esto que para establecer una planificación se debe estimar el tiempo que se empleará para la realización de este trabajo en función de las características del presente plan de estudios. En dicho plan de estudios se indica que la realización de este trabajo de fin de grado corresponde a 12 créditos ECTS, que equivalen a 300 horas de trabajo. Por lo tanto para realizar la planificación se debe repartir este tiempo entre las fases mencionadas anteriormente.

La fecha de comienzo del proyecto se fijó en el día 22 de enero de 2018. Teniendo en cuenta que se pretende presentar este proyecto en la convocatoria de junio, disponemos como fecha límite hasta el día 19 de junio de 2018 para finalizarlo. Dado que incluir esta fecha en los plazos del proyecto resulta un tanto arriesgado se pondrá de fecha límite el día 10 de junio de 2018. Además, se establecerá un margen de tiempo destinado a cubrir imprevistos que puedan tener un impacto negativo en el tiempo que se dedique a cada fase; en otras palabras, que el proyecto sufra desviaciones de tiempo con respecto a la planificación aquí propuesta.

El reparto de tiempo dedicado a cada fase es por tanto el siguiente:

Etapa	**Tiempo (h)**
Análisis	40h
Diseño	50h
Implementación	60h
Evaluación	40h
Documentación	80h
Margen	30h
TOTAL	300h

Tabla 2.1. PLANIFICACIÓN TEMPORAL DE ETAPAS

Una vez expuesto el reparto temporal de tareas se debe calcular en cuántos días finalizaremos cada una de ellas para proporcionar las fechas estimadas de inicio y finalización para cada fase. Para ello, y como se ha mencionado anteriormente, se sabe que el proyecto da comienzo el 22 de enero de 2018 y finaliza el 10 de junio de ese mismo año, por tanto,

el proyecto tiene una duración prevista de 139 días. Puesto que la persona encargada de realizar este proyecto trabaja de lunes a viernes no se han contado los festivos y fines de semana como no laborables, ya que era en estos días cuando se aprovechaba para avanzar más, por tanto, se han utilizado los 139 días íntegros para el cálculo de la planificación.

Puesto que se disponen de 300 horas reales para la realización del trabajo, se puede calcular el tiempo medio que se debe dedicar a la realización de cada etapa definida para este trabajo con el fin de completarlo sin sobrepasar la fecha final estimada.

$$\frac{H_{Totales}}{D_{Totales}} = \frac{300}{139} = 2{,}16 \quad horas/dia \tag{2.1}$$

Con las horas establecidas para cada fase tenemos que:

- Para la fase de Análisis necesitaremos unos 19 días para completar esta fase.
- Para la fase de Diseño necesitaremos unos 23 días para completar esta fase.
- Para la fase de Implementación necesitaremos unos 27 días para completar esta fase.
- Para la fase de Evaluación necesitaremos unos 19 días para completar esta fase.
- Para realizar la Documentación necesitaremos unos 37 días.
- Disponemos de 14 días de margen.

Así, a partir de estos cálculos de días, se ha realizado la planificación inicial del proyecto. Esta planificación inicial puede consultarse en la imagen del Anexo IV. Diagramas GANTT.

Resulta importante destacar que en esta planificación se ha optado por un enfoque optimista, ya que no se han tenido en cuenta posibles retrasos en las distintas fases del proyecto ni otros contratiempos posibles. Para ello mismo se ha destinado un margen de tiempo de dos semanas que, aunque no aparezca reflejado en el diagrama Gantt, está destinado a paliar los posibles efectos de los contratiempos que puedan surgir.

Por último, comentar que, aunque la documentación se ha dejado como última etapa para el final del proyecto, se refiere al hecho de redactar todo el documento de una manera más formal. No obstante, durante la realización de todo el proyecto se han ido documentando los pasos seguidos, problemas encontrados y otros aspectos que pudieran ser de interés.

2.3. Ejecución Final y Análisis de Costes

2.3.1. Planificación final

En este apartado se muestra la planificación final que se siguió para la realización del proyecto, resultado de tener que adaptar tiempo y esfuerzo a los diferentes contratiempos que iban surgiendo. El diagrama Gantt con la planificación final se muestra en la imagen Anexo IV. Diagramas GANTT

En primer lugar, es importante detallar en qué fases se produjeron dichos contratiempos y qué impacto tuvieron en el desarrollo normal del proyecto. Esta información se muestra en la siguiente tabla:

Etapa	Planificación inicial (días)	Planificación final (días)	Desviación (días)
Análisis	19	19	0
Diseño	23	26	+3
Implementación	27	33	+6
Evaluación	19	22	+3
Documentación	37	38	+1
TOTAL	125	138	+13

Tabla 2.2. DESVIACIÓN TEMPORAL SUFRIDA EN LA PLANIFICACIÓN

Como se puede observar en la tabla superior, al final de todo el proyecto se obtuve una desviación respecto a la fecha inicial estimada de 13 días. Los principales retrasos se obtuvieron en las etapas de Diseño, Análisis e Implementación. Aunque se explicará más adelante en el documento, estos retrasos se produjeron principalmente a las complicaciones surgidas a la hora de realizar el análisis de la información original para el diseño de la base de datos relacional. Aunque también hubo retrasos en la fase de evaluación, estos fueron debidos a la falta de conocimientos a la hora de realizar las consultas en MongoDB. A medida que fue pasando el tiempo fuimos ganando soltura y el desarrollo resultó más sencillo.

Por otro lado, se puede apreciar que la desviación más importante se produjo en la fase de implementación; esto fue debido a la cantidad de problemas que surgieron a la hora de realizar la inserción de los datos en Oracle. Este punto también se explica en detalle más adelante.

Por último, comentar que el retraso en la fase Documentación fue debido a la coincidencia de la realización de esta fase con el periodo de exámenes de la universidad, con lo que el tiempo disponible disminuyó ligeramente.

Como se ha observado, en el conjunto del proyecto se obtuvo un retraso total de 13

días respecto a la fecha inicialmente planteada. No obstante, gracias al tiempo de margen que se decidió dejar en la planificación inicial, la finalización del proyecto entró dentro de los plazos establecidos ya que el margen de tiempo establecido era de 14 días y se marcaba como fecha límite el día 10 de junio. A pesar de haber tenido un retraso de 13 días, la fecha de finalización se estableció en el día 8 de junio por lo que entra dentro de los plazos establecidos.

2.3.2. Cálculo de Costes

En este apartado se realiza el análisis del cálculo de costes que supondría la realización de este proyecto.

En primer lugar, se establece que el cálculo de estos costes se dividen en costes directos e indirectos. Los costes indirectos se han determinado como un 20 % de los costes directos, mientras que estos últimos están compuestos por los costes de personal y materiales que sen necesitado para la realización del proyecto.

Para comenzar se calcularán los costes de personal, para ello se ha tomado como referencia los sueldos medios presentados en la plataforma Jobtonic [1]. Los perfiles profesionales requeridos para este proyecto han sido los siguientes:

Perfil profesional	Sueldo bruto anual (€)	Coste/hora (€/h)	Coste total (€)
Analista de Datos	25.000	11,16	446,43
Arquitecto de Oracle	50.000	11,32	1116,07
Desarrollador	30.000	13,39	803,57
Jefe de proyecto	35.000	15,63	4656,25

Tabla 2.3. COSTES DE PERSONAL APLICADOS AL PROYECTO

Para realizar los cálculos, a partir del sueldo bruto anual se ha calculado el coste por hora de cada perfil dividiendo ese sueldo entre 14 pagas anuales para obtener el sueldo mensual. Este a su vez se ha dividido entre 4 semanas que tiene cada mes y entre 40 horas que se trabajan semanalmente.

Por otro lado, para calcular el coste total de cada perfil en el proyecto, se ha multiplicado el coste/hora de cada uno por el número de horas establecidas para cada etapa (Análisis: 40h, Diseño: 60h, Implementación: 60h).

Para calcular el coste total derivado del trabajo del jefe del proyecto se ha asumido que este perfil participa en el proyecto desde su comienzo hasta el final del mismo, por lo que hace falta conocer las horas de duración que ha tenido el proyecto; en el apartado

[1] http://espana.jobtonic.es/salary/26526/16078.html

anterior se calculó el número de días de duración del proyecto y así como el número de horas que se deberán trabajar por día para poder entregar el proyecto a tiempo. Por lo tanto, las horas de duración vendrán determinadas por los días totales multiplicado por el número de horas que se deben dedicar diariamente.

$$Das_{Totales} \cdot Horas_{Da} = 138 \cdot 2{,}16 = 298{,}08 \quad horas \tag{2.2}$$

Por tanto, se habrán dedicado 298 horas para completar el proyecto que se multiplicarán por el coste/hora del jefe de proyecto. Así, sumando el coste total de cada perfil profesional implicado en el proyecto obtenemos que el gato total de personal de proyecto es de 7022,32€ (siete mil veintidós como treinta y dos euros).

Para los gastos materiales se calcula su coste imputable en función de los días que se ha utilizado cada elemento.

Descripci	Coste (€)	% Uso	Dedicación	Vida útil (días)	Coste imputable
iMac	1.190	75	138	1825	67,5
Lenovo	500	25	138	1825	9.45
Oracle	0	50	20	1825	0
MongoDB	0	50	20	1825	0
Python	0	25	10	1825	0
TOTAL	**1.690**				**76.95**

TABLA 2.4. COSTES IMPUTABLES MATERIALES

A la vista de esta tabla resulta importante destacar que se han utilizado licencias académicas del software, las cuales no tienen ningún coste porque no se comercializa el resultado obtenido en el proyecto.

Por otro lado, la vida útil en días se ha calculado a partir del tiempo medio que tarda un equipo en perder su valor que está estimado en cinco años (1825 días) y el coste imputable se ha calculado con la fórmula de la amortización ya que no podemos cargar al proyecto el coste total de un equipo que se ha utilizado también fuera de ese proyecto. Por tanto, se calcula el valor imputable como:

$$Tiempo_{uso}/Tiempo_{vidaUtil}) \cdot Coste_{equipo} \cdot \ \%Uso \tag{2.3}$$

Una vez se han obtenido los costes de material y personal nos encontramos con que los costes directos ascienden a un total de 7099,27€ (siete mil noventa y nueve coma veintisiete euros).

Calculando el 20 % de esta cantidad obtenemos que los costes indirectos ascienden a un total de 1419,85€ (mil cuatrocientos diecinueve coma ochenta y cinco euros).

Si sumamos las cantidades de costes directos e indirectos obtenemos que el precio sin IVA del proyecto ha sido 8519,12€ (ocho mil quinientos diecinueve coma doce euros).

Finalmente, aplicando un IVA del 21 % sobre esta cantidad, obtenemos que el coste total del proyecto asciende a 10308,14€ (**DIEZ MIL TRESCIENTOS OCHO COMA CATORCE EUROS**).

3. ESTADO DEL ARTE

En este capítulo se pretende hacer una presentación de los distintos paradigmas de almacenamiento de datos existentes en la actualidad, haciendo un repaso de sus principales características, así como en qué ocasiones es recomendable usar un tipo u otro.

3.1. Sistemas de almacenamiento tradicionales

Desde los inicios de la humanidad, el ser humano siempre ha querido almacenar información; las antiguas civilizaciones también guardaban información escribiéndola en medios más rudimentarios como las tablillas o los pergaminos. Con el paso del tiempo estas necesidades de almacenar información fueron cambiando. Sin embargo, no fue hasta que apareció la informática cuando se produjo las necesidades de almacenamiento de información cambiaron completamente y comenzaron a surgir nuevos medios.

Uno de los primeros sistemas utilizados a la hora de almacenar información fue el sistema de ficheros. Nacido en la década de 1950 con las cintas magnéticas [2], estaba compuesto por un conjunto de programas que operaba sobre una serie de ficheros en los que se almacenaba la información. Lo que en un principio pareció una buena forma para almacenar datos lógicos, pronto surgieron los primeros problemas. Entre estos problemas se encontraban los datos redundantes, distintos formatos para los mismos datos o programas muy dependientes de la estructura de los ficheros, lo que suponía un gran inconveniente a la hora de realizar cualquier cambio [3].

Debido a estos problemas, surgió la idea de crear almacenes de datos independientes de los programas que los manipulaban, para que así cualquier cambio que se acometiera en uno u otro no afectara a los dos.

Todo ello llevó a la aparición de las bases de datos a finales de los años sesenta y principio de los setenta con los trabajos del grupo CODASYL y el desarrollo del modelo relacional [2]. Estas bases de datos, aunque en un principio eran muy básicas y ofrecían pocas posibilidades, con el paso del tiempo y la aparición de nuevas necesidades, han ido evolucionando hasta llegar a los actuales sistemas gestores de bases de datos.

A continuación, se explicarán detalladamente los sistemas de almacenamiento jerárquico, en red, orientado a objetos, declarativo y relacional.

3.1.1. Sistema de Almacenamiento Jerárquico

Es un tipo de sistema de gestión de bases de datos en el cual los datos son organizados en una estructura jerárquica similar a un árbol. También se permiten las relaciones entre nodos hermanos, aunque en este caso la estructura resultante se asemeja más a un grafo

dirigido.

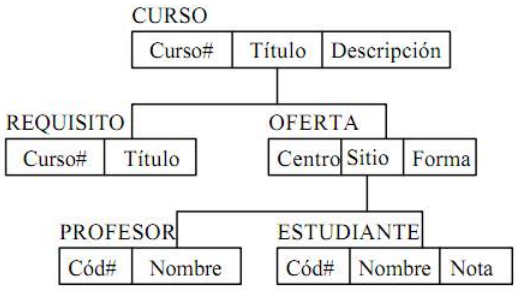


Fig. 3.1. Estructura sistema jerárquico [4]

Este modelo jerárquico facilita las relaciones 1:N, pero el principal problema de las mismas es que son unidireccionales, es decir, son relaciones hijo-padre pero no en sentido contrario. Esto implica que no se permite acceder a los nodos hijos sin haber visitado previamente los nodos padres. Una consulta en sentido contrario requeriría una búsqueda secuencial por todos los registros de la base de datos [4]. Por otro lado, una de las grandes limitaciones de este modelo es la incapacidad para representar de una manera eficiente la redundancia de datos ya que puede ocurrir que se modifique únicamente una de las instancias y los datos queden inconsistentes.

El principal SGBD que implementa este modelo jerárquico es el IMS de IMB.

3.1.2. Sistema de Almacenamiento en Red

Este tipo de modelado es una evolución del modelo jerárquico en el cual se permite que un mismo nodo tenga varios nodos padres, solucionando así de una manera eficiente el problema de la redundancia de datos [5]. Este modelo de datos utiliza principalmente dos conceptos: registros y conjuntos. Los registros tienen campos, mientras que los conjuntos definen las relaciones entre los registros.

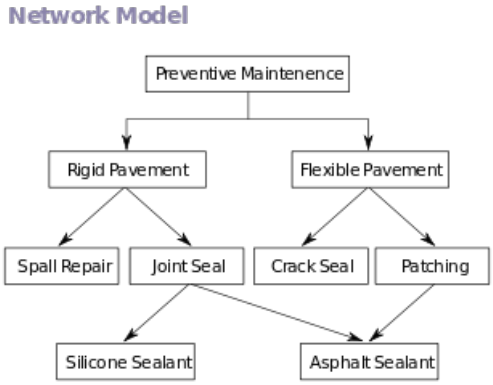


Fig. 3.2. Estructura almacenamiento en red [5]

Uno de los principales problemas de este tipo de bases de datos reside en la dificultad para administrar la información dada la complejidad de las relaciones. También es importante destacar que en el modelado en red no existen restricciones de integridad ni referenciales.

3.1.3. Sistema de Almacenamiento Orientado a Objetos

Este tipo de bases de datos surgen de la necesidad de almacenar objetos cada vez más complejos dado el creciente uso del paradigma de programación orientado a objetos. Una de las principales características de estas bases de datos es la potencia que proporcionan a la hora de definir la estructura de objetos complejos, así como las operaciones que se pueden aplicar sobre esos objetos. Mediante el uso de estas bases de datos no hace falta implementar capas de persistencia adicionales para gestionar dichos objetos [6].

Este modelo es utilizado cuando se tiene un sistema que requiere un buen rendimiento a la hora de manipular tipos de datos complejos. No obstante, algunos aspectos como la carencia de estándares o el hecho de que las bases de datos relacionales implementan herramientas de soporte a objetos hacen que en la práctica se utilicen más estas últimas [7].

3.1.4. Sistema de Almacenamiento Declarativo

Este modelo es usado principalmente para bases de conocimiento, en las cuales las consultas se realizan sobre cantidades enormes de información y en ocasiones requieren de sistemas distribuidos para llevar a cabo de una manera más eficiente las operaciones que se realicen.

Dentro de las bases de datos declarativas se pueden diferenciar dos grupos:

- Modelo deductivo: aunque este modelo también está integrado por un conjunto de tablas, estas son radicalmente distintas a las que se pueden encontrar en un modelo relacional. En este caso una tabla no se considera como un conjunto de tuplas, sino como un conjunto de predicados lógicos [4]. Una base de datos deductiva es capaz de determinar la veracidad o falsedad de y bajo qué circunstancias, así como deducir relaciones indirectas entre los datos que se almacenan. De esta forma, en las bases de datos deductivas la información se puede indicar como hechos o como secuencias de deducción, lo que supone un gran avance para las aplicaciones en el campo de la Inteligencia Artificial, entre otras.

- Modelo funcional: estas bases de datos extienden el modelo de programación funcional, según el cual todo objeto computacional debe comportarse como una función y por lo tanto ante las mismas entradas debe responder siempre con las mismas salidas [4].

Por último, mencionar que estas bases de datos utilizan un lenguaje puramente declarativo para las operaciones, en el cual el usuario no es consciente de los métodos de búsqueda que se realizan internamente y la forma de manejar los datos también es muy distinta. Este lenguaje declarativo es un lenguaje no procedural ya que el usuario especifica qué información quiere pero no cómo.

3.1.5. Sistema de Almacenamiento Relacional

En el año 1970 el doctor Edgar F.Codd, científico informático de IBM, presentó un escrito titulado 'A Relational Model of Data for Large Shared Data Banks' [8], lo que supuso el inicio del campo de las bases de datos relacionales. El modelo relacional tiene su base teórica en la teoría de conjuntos y la lógica de predicado de primer orden. Este modelo pronto comenzó a llamar la atención debido a su simplicidad y fundamentación matemática.

En el modelo relacional una base de datos se representa como una colección de relaciones, representadas por tablas. Aplicando este modelo, una fila recibe el nombre de tupla, la cabecera de la columna recibe el nombre de atributo y el nombre de una tabla se conoce como relación. Cada tupla es identificada de manera unívoca mediante un campo único denominado clave primaria y las relaciones entre las distintas tablas se establecen a través de estos identificadores, que cuando se refieren a otra tabla reciben el nombre de clave ajena.

Fig. 3.3. Sistema de almacenamiento relacional

De esta manera, tenemos que uno de los principales puntos a favor de las bases de datos relacionales es que poseen herramientas que permiten evitar la duplicidad de registros así como mecanismos para asegurar la integridad referencial de la base de datos ayudando a la consistencia y a la precisión de los datos.

Otro punto fuerte de las bases de datos relacionales, y posiblemente una de las razones del éxito comercial de este modelo, es que poseen un lenguaje de consulta estructurado conocido como SQL, siglas de Structured Query Language. Este lenguaje fue diseñado por IBM Research para un sistema gestor de bases de datos propio que con el tiempo se fue convirtiendo en un estándar. La ANSI en colaboración con la ISO sacaran la primera

especificación de este lenguaje en el año 1986, recibiendo el nombre de SQL1 o SQL-86 [8]. Desde entonces este estándar ha ido creciendo con modificaciones y mejoras y la última versión actual fue lanzada en el año 2016. Mediante SQL se puede definir cómo manipular datos, así como temas de seguridad y autorización, o definir restricciones de integridad.

Como puntos negativos del modelo relacional se puede mencionar que a medida que aumenta el tamaño de la base de datos la eficiencia de las operaciones sobre la misma también decae. Esto hace recurrir a otras opciones como dividir las tablas más grandes en varios fragmentos u optar por escalar el sistema. Otro punto negativo importante a mencionar es la necesidad de un experto que realice un buen diseño de la base de datos, ya que un mal diseño puede afectar negativamente al rendimiento de la base de datos, así como dar lugar a anomalías o inconsistencias a la hora de hacer actualizaciones.

A pesar de todo, el modelo relacional es el más extendido en el mercado y las bases de datos relacionales las más utilizadas en todo tipo de ámbitos. Algunos ejemplos de bases de datos relacionales son: Oracle [2], SQL Server [3] o MySQL [4].

3.1.6. Sistemas de Almacenamiento Distribuido

Aunque las bases de datos relacionales son eficientes con datos muy bien estructurados, se puede dar la situación de que la base de datos crezca demasiado en tamaño y que las consultas y operaciones con determinadas tablas grandes se vuelvan ineficientes y consuman muchos recursos y tiempo. Así, nacieron las bases de datos distribuidas; mediante las cuales se consigue distribuir la información entre distintos nodos de una única red que funcionan conjuntamente a la hora de realizar las operaciones pero actúan como una única base de datos [9]. A la hora de diseñar el esquema de una base de datos distribuida, se puede optar porque los nodos actúen como réplicas de alguna tabla o, por otro lado, fragmentar las tablas más grandes y dividirlas entre los distintos nodos [10].

Por lo tanto, tenemos que las bases de datos distribuidas tienen por finalidad, por un lado, la compartición de los datos entre los distintos nodos; y por otro lado la optimización de consultas mediante la realización de estas a través de distintas máquinas.

A pesar de ser estructuras con redes de alta velocidad y de alto rendimiento, tienen algunos inconvenientes como la dificultad de realizar un buen diseño, la dependencia física de la red de comunicación entre los distintos nodos o la complejidad de la administración de la base de datos [9].

[2] https://www.oracle.com/es/index.html
[3] https://www.microsoft.com/es-es/sql-server/sql-server-2016
[4] https://www.mysql.com/

3.1.7. Sistemas de Almacenamiento Paralelo

Cuando una base de datos crece demasiado puede darse la situación de que las consultas y operaciones con la misma se vuelvan ineficientes debido a las características del servidor. En esta situación, si se necesita un alto rendimiento de la base de datos, una posible alternativa a una base de datos distribuida es optar por una base de datos paralela. La principal diferencia entre ambos sistemas de datos es que las bases de datos paralelas tienen un único esquema centralizado en un nodo y un punto de inicio para la ejecución de las operaciones [11].

A la hora de diseñar una base de datos paralela se puede optar por tres arquitecturas: de discos compartidos (cada procesador tiene su memoria principal y comparten disco duro), todo se comparte (los procesadores acceden al mismo espacio de direcciones y al mismo disco duro) y nada se comparte (cada procesador tiene su propia memoria principal y memoria secundaria y se comunican mediante paso de mensajes) [12].

Así, una base de datos paralela se encuentra formada por un servidor con varios procesadores entre los que se dividen las operaciones; aprovechando las características de la computación paralela y mejorando la velocidad de procesamiento y de entrada/salida.

3.2. Nuevas tendencias

Aunque estos sistemas tradicionales tienen multitud de aplicaciones y de utilidades, como ya se ha mencionado anteriormente surgieron a partir de unas necesidades específicas que surgieron con la aparición y evolución de la informática y de las necesidades de almacenar cada vez más información que esta provocaba.

Así, con el paso del tiempo estas necesidades han ido evolucionando y ahora las nuevas necesidades necesidades se centran en disponer de una gran cantidad de información para realizar sobre ella tareas de análisis y a partir de este análisis tomar mejores decisiones.

Con estas nuevas necesidades las los sistemas de almacenamiento tradicionales se han quedado atrás ya que el rendimiento de este tipo de sistemas no era todo lo eficiente como se necesitaba. Así, han surgidos nuevos sistemas y nuevos paradigmas de almacenamiento que permiten trabajar con una gran cantidad de datos y realizar operaciones de todo tipo sobre ella sin que el rendimiento se vea afectado. En este capítulo se realiza una introducción a los diferentes paradigmas que han ido apareciendo y para qué se suele utilizar cada uno.

3.2.1. Sistemas de almacenamiento analítico

Dado el nivel competitivo que se ha alcanzado por parte de las empresas en los últimos tiempos, se ha hecho necesario el desarrollo de nuevas estrategias de gestión que lleven a un mejor uso de la información que ayude a la toma de decisiones; son los denominados sistemas de almacenamiento analíticos

Dadas las carencias de los sistemas tradicionales a la hora de resolver estas necesidades, surgió el término Data Warehouse. Este término apareció a finales de los años 80 y fue acuñado por Bill Inmon en el año 1996, quien lo definió como una “colección de datos orientados al tema, integrados, no volátiles, historiados, organizados, para el apoyo de un proceso de ayuda a la decisión” [13]. Así, un Data Warehouse es una tecnología que proporciona una herramienta competitiva a las empresas, ya que permite integrar, organizar y almacenar datos para posteriormente analizarlos; y añadiéndoles una variante temporal. Esta variante temporal se modela como una dimensión más del negocio, y permite hacer análisis multidimensionales.

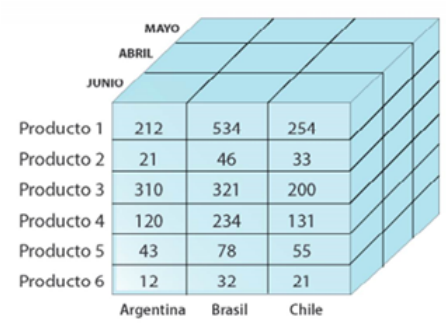


Fig. 3.4. Estructura cubo Data Warehouse [14]

Como se puede apreciar en la imagen superior, a través de esta estructura tridimensional, además de consultar el número de productos que se han vendido en un determinado país o en qué país se ha vendido un producto, se puede saber hacer un análisis temporal de esa misma información.

3.2.2. Sistemas de almacenamiento espacio-temporal

Aunque tradicionalmente se ha trabajado con información estructurada, gestionada mediante las bases de datos relacionales, en los últimos tiempos ha surgido la necesidad de almacenar y trabajar con información no estructurada y dinámica, provocando también la aparición de nuevas técnicas que nos permitan trabajar de una manera eficiente con este tipo de información. En ocasiones, es necesario almacenar objetos que tiene asociada información espacio-temporal, es decir, objetos cuya posición espacial o forma cambien en función del tiempo; de esta necesidad surgen las bases de datos espacio-temporales [15].

Este sistema de almacenamiento espacio-temporal permite trabajar con información con una estructura compleja y con algún atributo espacial. Estos sistemas proveen mecanismos eficientes para procesar consultas sobre información con este tipo de atributos, ya que tratan el espacio y el tiempo como una dimensión más del objeto.

A pesar de que este tipo de sistemas proveen mecanismos para tratar las operaciones de una manera eficiente, estas bases de datos suelen tener un gran tamaño y debido a la complejidad de este tipo de datos suelen implicar un coste de procesamiento bastante alto [16].

El principal ámbito de aplicación de este tipo de sistemas de almacenamiento es en sistemas de información geográfica.

3.2.3. Sistemas de almacenamiento NoSQL

El término NoSQL (Not Only SQL) engloba a todas las tecnologías de almacenamiento estructurado que no cumplen el esquema relacional. Este concepto surgió a finales de

los años 90 y se refería a una base de datos relacional de código abierto que no utilizaba como lenguaje de consulta SQL. A pesar de existir desde los años noventa, el término NoSQL no se popularizó hasta el año 2009 cuando Johan Oskarsson organizó un evento para tratar las bases de datos distribuidas de código abierto no relacionales, llamándolas NoSQL [17].

Este concepto de NoSQL surge de la necesidad de las grandes empresas de gestionar las grandes cantidades de información que poseen, lo que ha llevado a que surjan nuevas arquitecturas de almacenamiento de información, buscando conseguir un alto rendimiento, escalabilidad y distribuidas.

Aunque este tipo de bases de datos surgieron de unas necesidades muy concretas, la difusión de las mismas y el empeño continuo en hacerlas más amigables a desarrolladores acostumbrados a trabajar con SQL ha provocado que se empiecen a usar en proyectos de menor tamaño, lo que ha llevado a que convivan con las bases de datos tradicionales independientemente del volumen de datos a gestionar.

3.2.4. Tipos de bases de datos NoSQL (taxonomía)

A continuación, se presentan los distintos tipos de bases de datos NoSQL existentes. Aunque debido a la gran cantidad de soluciones NoSQL (actualmente existen más de 250 sistemas NoSQL) existen varios criterios de clasificación, cuando se habla de categorías se pueden destacar cuatro.

3.2.4.1. Bases de datos Clave-Valor

De las bases de datos NoSQL las de clave-valor seguramente sean las que siguen el modelo más simple. A pesar de ello, Este tipo de bases de datos prometen un rendimiento excelente con volúmenes de datos muy grandes, a cambio de renunciar a algunas funcionalidades, como la verificación de la integridad de los datos, delegando la implementación de estas funciones a la aplicación cliente [18].

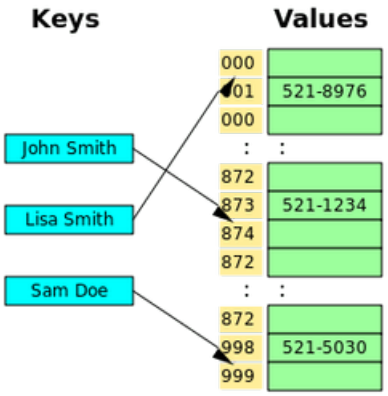


Fig. 3.5. Esquema clave-valor [19]

Este tipo de bases de datos consisten en un mapa o diccionario (DHT) en el cual se asigna una clave única a un valor. La operación de encontrar el valor asociado a una clave se denomina “lookup” (indexación), y la relación entre una clave y su valor se denomina correlación.

Debido a la naturaleza poco estructurada de las bases de datos clave-valor este modelo favorece la escalabilidad sobre la consistencia [20]. Se recomienda su uso en casos donde se necesite velocidad en las consultas o se tienen muchos datos con estructura simple que requieren ser procesados una y otra vez y tienen valores cambiantes.

Aunque el modelo clave-valor es simple, las consultas de rango no son directas y, de modo general, resultan muy complejas de ejecutar. Este tipo de bases de datos han aumentado de popularidad debido en gran parte a los sistemas basados en Cloud Computing.

Algunos ejemplos de este tipo de bases de datos son DynamoDB [5] o Redis [6].

3.2.4.2. Bases de datos columnares

En este tipo de bases de datos se almacena la información por columnas, en lugar de por filas como en el modelo relacional. Al almacenar la información en columnas todos los valores de un campo pueden ser accedidos a la vez, lo que las convierte en especialmente eficaces a la hora de ejecutar consultas analíticas [18]. Por otro lado, cada columna es almacenada de forma contigua en un lugar separado en disco, utilizando unidades de lectura grandes para facilitar la búsqueda de varias columnas; reduciendo así notablemente los requisitos globales de E/S del disco y reduciendo la cantidad de datos que hay que cargar del mismo.

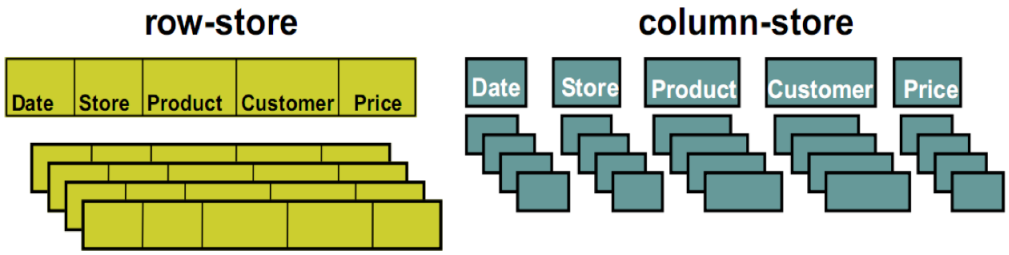


Fig. 3.6. Esquema base de datos columnar [21]

Para mejorar la eficiencia de las operaciones de lectura, se suelen utilizar esquemas de compresión ligera de los datos cuando es posible para hacer así que ocupen menos [21]. Este punto en ocasiones puede ocasionar un impacto negativo para el rendimiento de la base de datos, ya que se debe realizar una descompresión de los datos a la hora de realizar lecturas. Otros sistemas utilizan como alternativa menos compresión de datos o almacenan varias versiones de los datos comprimidos, afectando a otros beneficios de este

[5] https://aws.amazon.com/es/dynamodb/
[6] https://redis.io/

tipo de almacenamiento. En este aspecto el hardware de las máquinas disponibles puede suponer un punto diferencial.

Este tipo de bases de datos se recomiendan para grandes almacenes de datos (del orden de Petabytes) en los que las operaciones de lectura son abundantes y las de escritura más escasas, o es necesario realizar muchas operaciones con los atributos de las entidades, como en los campos Business Intelligence o Data Analytics [21].

Algunos ejemplos de este tipo de bases de datos son Cassandra[7] o HBase[8].

3.2.4.3. Bases de datos documentales

Este tipo de bases de datos están diseñadas para gestionar información orientada a documentos o datos semi-estructurados almacenados en distintos formatos como podrían ser JSON, XML, o YAML [22]. Una de las principales diferencias con las bases de datos relacionales es que no es necesario definir un esquema que deban seguir todos los documentos, sino que son de estructura libre [18]. Este aspecto ofrece a los desarrolladores y administradores de bases de datos una mayor flexibilidad a la hora de organizar y almacenar los datos de las aplicaciones, así como una reducción del almacenamiento requerido para valores opcionales.

En la mayoría de bases de datos documentales el formato utilizado por los documentos es JSON (JavaScript Object Notation). Este formato es soportado por varios lenguajes de programación actuales y es más compacto y legible que cualquier otro; lo que lo hace muy fácil de usar tanto por humanos como por ordenadores. Otra característica importante es que los datos se pueden modelar de una manera similar a la programación orientada a objetos; dentro de un documento se pueden introducir arrays o incluso subdocumentos que contengan dentro más información.

```
{
    "nombre": "Miguel",
    "apellido": "De Arriba",
    "dirección": {
        "calle": "",
        "número": "",
        "CP": ""
    },
    "amigos": ["Luis","Pedro","Ángel"],
    "profesion": "estudiante"
}
```

Fig. 3.7. Estructura documento

Con los documentos se consigue almacenar y recuperar todos los datos relacionados como una sola unidad, lo cual puede suponer grandes ventajas de rendimiento y escalabilidad.

[7] http://cassandra.apache.org/
[8] https://hbase.apache.org/

Algunos ejemplos de este tipo de bases de datos son MongoDB[9] o CouchDB[10].

3.2.4.4. Bases de datos orientadas a grafos

Estas bases de datos se apoyan en la rama de las matemáticas de la teoría de grafos desarrollada por Leonhard Euler en el siglo XVIII [23]. Así, tenemos que los vértices representan las entidades, mientras que las aristas representan las relaciones y propiedades de las mismas.

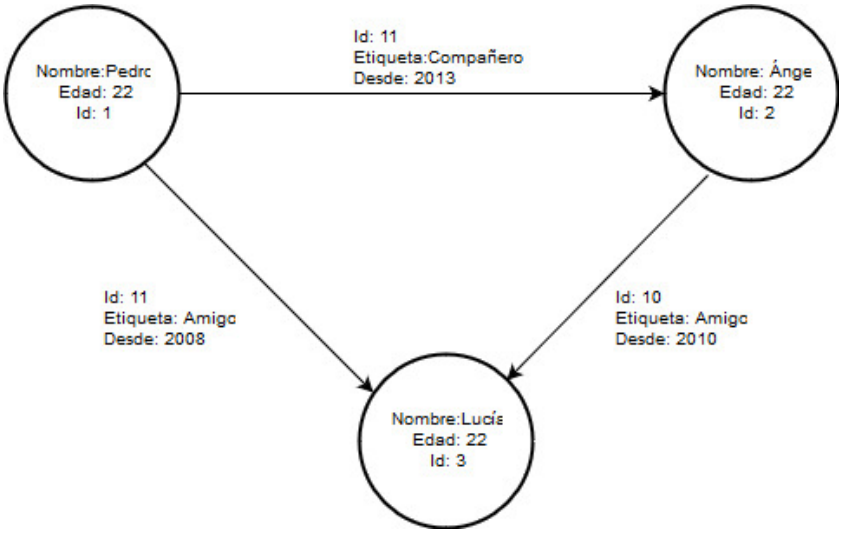


Fig. 3.8. Base de datos orientada a grafos

Dada la capacidad de añadir nuevos nodos o relaciones sin necesidad de alterar el comportamiento de la aplicación, estas bases de datos son recomendadas para sistemas dinámicos en los que la lógica de negocio cambie con frecuencia y exista la necesidad de adaptación constante.

Por otro lado, el rendimiento en estas bases de datos tiende a permanecer constante [24] independientemente del número de nodos o relaciones que tenga el grafo. Esto ocurre porque a la hora de realizar una consulta el recorrido se comienza desde un segmento de datos del grafo, recorriendo los nodos y aristas que satisfacen la consulta. Así, el tiempo de ejecución no dependerá del tamaño total de grafo, sino del tamaño del subgrafo que se recorre en la consulta. Estas bases de datos son eficientes para realizar consultas en las que existen relaciones de proximidad entre datos, pero no para ejecutar consultas sobre la totalidad de los datos.

Actualmente los principales usos de estas bases de datos son en redes sociales, algoritmos de recomendaciones o para controles de acceso. Algunos ejemplos de estas bases de datos son Neo4j[11] o InfiniteGraph[12].

[9] https://www.mongodb.com/
[10] http://couchdb.apache.org/
[11] https://neo4j.com/
[12] http://www.objectivity.com/products/infinitegraph/

3.3. Elección del mejor sistema de almacenamiento

Como se ha visto, dada la cantidad de diferentes sistemas de almacenamiento existentes en la actualidad; es normal que a una empresa le pueda resultar complicado decantarse por un sistema u otro a la hora de implantar una base de datos. No obstante, esta puede ser una decisión crítica para el negocio, ya que puede intervenir directamente en el buen desarrollo del mismo.

Son varios los casos de empresas que por desconocimiento se decantan por un sistema de base de datos porque han oído que está de moda o porque alguna empresa de la competencia lo está utilizando y al poco tiempo tienen que cambiar a otro que les ofrezca otras características distintas que se ajusten más a lo que necesitaban. También sucede incluso que, en ocasiones, se pierden datos en el proceso de transformación y migración de un sistema a otro. En muchas ocasiones es tan importante tener un buen diseño de base de datos como hacer una buena elección del sistema que se va a utilizar. Si un mal diseño de la base de datos puede provocar que los usuarios accedan con dificultades a la información y provocar inconsistencias o información falsa, una mala elección del sistema de base datos puede tener una repercusión directa en el rendimiento de la misma y por tanto del negocio.

Todo ello se puede evitar con un previo análisis de la información de la que se dispone y cuáles son las necesidades que se tienen. Así, en función de la información que se tenga, si está estructurada o no, y de lo que se quiera priorizar (tiempo de respuesta del sistema, optimización de espacio, análisis analítico de la información, respuestas exactas/aproximadas,...) se deberá optar por un sistema u otro.

En este trabajo de fin de grado se busca poner de manifiesto la importancia de realizar este análisis previo a la elección del sistema de base de datos que se vaya a utilizar mediante la implementación de una serie de consultas en un sistema relacional y en un NoSQL para estudiar en profundidad la eficiencia de consultas complejas y ver cómo se comporta cada sistema.

3.4. Entorno socio-económico

Actualmente vivimos en un mundo conectado a Internet, todo el mundo posee algún móvil, ordenador o tablet con el conectarse a Internet. Es tal el número de dispositivos conectados que en el año 2017 hubo 8400 millones de dispositivos conectados, cifra que se espera alcance los 20400 millones en el año 2020, según la firma de análisis Gartner [25]. Este crecimiento tan elevado del número de dispositivos, así como de la información generada, se debe principalmente al surgimiento de nuevos conceptos, como IoT o Big Data.

3.4.1. Big Data

El término Big Data hace referencia a la ingente cantidad de datos que, debido a su tamaño y sus características no pueden ser tratados con las herramientas convencionales con las que se venían tratando hasta ahora. Sin embargo, la principal características de Big Data no son la cantidad de datos que se gestionen o la naturaleza de los mismos, sino el uso que se hace de ellos; mediante técnicas de Big Data se pueden obtener nuevos datos que ayuden a las empresas a tomar mejores decisiones. Aunque este término está sujeto a una cierta polémica por discernir qué puede considerarse como Big Data y qué no, todos los investigadores y profesionales coinciden en que el término tiene cuatro características básicas llamadas “Las 4 V’s del Big Data", aunque este número ha ido incrementándose con el paso del tiempo [26].

A día de hoy, Big Data es utilizado en multitud de campos como pueden ser finanzas o deportes. También es utilizado en otros ámbitos más peculiares como puede ser predecir un crimen en una zona [27] o ayudar al actual presidente de los EE.UU. Donald Trump a ganar las elecciones [28].

3.4.2. Internet of Things (IoT)

Internet of Things (en adelante IoT) hace conexión entre objetos cotidianos y máquinas con acceso a la red. El término se utilizó en público por primera vez en el año 2009 por el profesor del MIT (Massachusetts Institute of Technology) Kevin Ashton, aunque él mismo ya utilizaba este concepto en varias investigaciones en el año 1999 [29].

IoT es considerado como la gran revolución en Internet tal y como lo conocemos hasta ahora. Mediante esta tecnología se consigue que objetos que antes trabajaban de manera aislada ahora puedan trabajar junto a otros dispositivos mediante redes. Así, tenemos que IoT es aplicable a sensores que miden la contaminación en las ciudades y permiten tomar medidas más adecuadas o gestionar el tráfico de las grandes ciudades en tiempo real de una manera más eficiente que los tradicionales semáforos. No obstante, IoT no está exento de polémica, especialmente en la carencia de seguridad de los dispositivos [30].

3.5. Marco regulador

Al estar tratando con datos, ya sean de sensores de IoT de cualquier tipo, o datos particulares de una empresa, se debe conocer cuál es la legislación que afecta directamente a esos datos y a su cuidado y protección. Concretamente, habrá que prestar especial atención a dos regulaciones: a nivel nacional la LOPD (Ley Orgánica de Protección de Datos) y a nivel europeo la nueva RGPD (Reglamento General de Protección de Datos).

3.5.1. LOPD

La Ley Orgánica de Protección de Datos fue aprobada en diciembre del año 1999 y tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar [31]. Esta ley de obligado cumplimiento para todas las empresas y organismos públicos que guarden en soportes físicos o electrónicos información relativa a personas físicas. Asimismo, se indica que la empresa u organismo encargado de recoger la información debe facilitar y garantizar a los ciudadanos ejercer libremente los denominados derechos ARCO (acceso, rectificación, cancelación y oposición) sobre sus datos.

Por otro lado, se establece que existen datos que por su relevancia e importancia para la privacidad deben ser tratados de manera especial; son los llamados datos sensibles o datos especialmente protegidos. Se consideran datos sensibles todos aquellos que hacen referencia a la intimidad, libertades públicas o derechos fundamentales de las personas (ideología, religión, creencias,...). Con la entrada en vigor de la RGPD la LOPD ha sufrido una serie de modificaciones en este aspecto y con la nueva norma se prohíbe por regla general el tratamiento de este tipo de datos salvo que exista un consentimiento explícito por parte del interesado y se den una serie de circunstancias [32].

Las sanciones por el incumplimiento de esta ley pueden ir desde los 900€ hasta los 600.000€ dependiendo de la calificación de la infracción (leve, grave o muy grave).

3.5.2. RGPD

El 25 de Mayo de 2016 entró en vigor el Reglamento General de Protección de Datos (RGPD), aunque no comenzará a aplicarse hasta el 25 de Mayo de 2018. Se deja ese periodo de tiempo para que las empresas y organizaciones puedan adaptarse para el momento en el que comience a aplicarse dicho reglamento. Es la primera norma de este tipo que afecta a todos los países de la Unión Europea, unificando los derechos y obligaciones de todos los ciudadanos de la Unión y de las empresas que operan en la misma independientemente del país en el que tengan establecida su sede (dentro o fuera de la Unión) o de la actividad que realicen.

Mediante esta nueva norma se concede un mayor control a los ciudadanos sobre los

datos que las empresas recogen, ya que se incrementan los derechos de los mismos a decidir cómo quieren recibir información de las empresas y cómo quieren que su información sea tratada. En el apartado de datos sensibles, además de los ya mencionados en la LOPD, se añaden los datos biométricos y genéticos. Por último, también se establece el derecho al olvido por parte de los ciudadanos, por el cual podrán solicitar la eliminación de toda su información en redes sociales y/o buscadores [33].

Las sanciones por el incumplimiento de esta norma podrán oscilar entre los 10 millones de euros o el 4 % del volumen de negocio total anual del ejercicio financiero anterior, dependiendo de cuál sea la de mayor cuantía entre las dos.

4. ANÁLISIS

A lo largo de este capítulo se explica el problema que se aborda en el presente Trabajo Fin de Grado, así como las tecnologías software que se han utilizado en el mismo y las distintas etapas que se han seguido.

4.1. Descripción del problema

Como ya se ha comentado anteriormente, las bases de datos NoSQL nacieron debido al surgimiento de nuevas necesidades, como almacenar gigantescas cantidades de datos para posteriormente analizar esa información con el objetivo, por ejemplo, de calcular correlaciones entre informaciones que aparentemente no guardaban ninguna relación. Esta nueva necesidad nace de las nuevas tecnologías utilizadas y por la forma en que actualmente Internet es accesible a todo el mundo, motivado todo ello por la aparición de las redes sociales o aplicaciones móviles. Si nos fijamos en los datos, según el portal estadísticas Statista, en el año 2017 se realizaron más de 3,8 millones de búsquedas en YouTube, se suben más de 65.000 fotos a Instagram y se envían más de 29 millones de mensajes en WhatsApp por minuto [34].

Fig. 4.1. 1 minuto en Internet [34]

A la vista de estos datos, se hace evidente la necesidad de poder tratar la información adecuadamente para intentar sacar el máximo beneficio o alguna ventaja respecto a

la competencia. Ante este escenario muchas empresas se ven en la necesidad de elegir qué tipo de base de datos utilizar, si decantarse por una base de datos relacional o bien optar por una solución NoSQL. Todo ello lleva a la aparición de nuevas técnicas y nuevas tecnologías para buscar obtener el máximo provecho de toda esta situación.

En este contexto, el problema surge a la hora de tener que elegir una base de datos para almacenar la información. Sucede que en ocasiones la empresa interesada en montar un servidor de base de datos no sabe qué entorno elegir, si un sistema relacional o NoSQL. Muchas veces se elige optando por lo que está de moda en el momento o lo que se ha escuchado a la competencia, todo ello sin hacer un estudio previo de lo que ofrece cada entorno, ni de la información que se quiere almacenar y para qué se quiere utilizar (Big Data, cálculos estadísticos, datos relacionados,...).

Como ya se ha visto en la sección 3.1.5. Sistema de Almacenamiento Relacional, las bases de datos relacionales funcionan muy bien con datos estructurados o si entre los datos almacenados existen relaciones. Sin embargo, a la hora de almacenar gran cantidad de información y poco estructurada pierden eficiencia y no permiten procesar los datos adecuadamente. Esta situación se debe sobretodo a que las bases de datos tradicionales están diseñadas para trabajar con una cantidad de datos limitada, a partir de la cual se hace muy costoso continuar dando respuesta a las peticiones que se realicen sobre ella debido a que se debe invertir en hardware más potente para que la eficiencia no decaiga. Esto ligado a que es necesario que los datos dispongan de estructura, hace que para procesar información desestructurada en grandes cantidades se deban buscar alternativas a las bases de datos tradicionales.

Este aspecto no tiene por qué significar que ahora únicamente se deban utilizar bases de datos NoSQL en detrimento de las bases de datos relacionales tradicionales. Actualmente son multitud las empresas siguen utilizando bases de datos relacionales que almacenan sus datos en bases de datos relacionales, convirtiendo éstas en una de las herramientas más utilizadas. No obstante, sí es cierto que en algunos casos concretos optar por un sistema NoSQL puede suponer una gran ventaja a la hora de manipular los datos. En cada caso se deberá ver la naturaleza de los datos y las operaciones que se necesitan realizar sobre ellos y analizar las ventajas y desventajas de cada tecnología. En función de todo ello se optará por un sistema relacional o NoSQL.

En este TFG se pretende hacer una comparativa de rendimiento entre una base de datos relacional y una NoSQL para ver la eficiencia de cada una ante una serie de consultas de diferente complejidad. Esta comparativa será un factor más a valorar a la hora de elegir entre ambas tipologías de bases de datos para el dominio concreto que se estudia y dominios de características similares.

4.2. Entorno operacional

Una vez vistos los distintos sistemas de almacenamiento existentes y presentadas las principales características de cada uno de ellos se han seleccionado dos de ellos para realizar la comparativa de rendimiento en el presente trabajo. Resulta importante destacar que dentro de los sistemas tradicionales, en este capítulo únicamente se van a tener en cuenta los sistemas relacionales dado que son los más utilizados en la actualidad y por lo tanto los que tienen una mayor relevancia.

En primer lugar, dado que en la mayoría de los sistemas de bases de datos resulta esencial la consistencia y la disponibilidad de la información, es de gran importancia analizar los distintos sistemas que cumplen este aspecto. La categorización que se le da a un tipo de base de datos tradicional o NoSQL proviene directamente de la aplicación del denominado Teorema CAP.

Este teorema nace en el año 2000 de la mano de Eric Brewer (profesor de la universidad de Berkeley en California) que estableció la conjetura de que los servicios de almacenamiento de datos no pueden asegurar de forma conjunta las propiedades Consistencia (C), Disponibilidad (A) y Tolerancia a fallos (P) [35].

Posteriormente, en el año 2002, los trabajadores del MIT Seth Gilbert y Nancy Linch publicaron una demostración formal de la conjetura de Brewer, convirtiéndola en un teorema [36]. Aunque esta demostración ha sido criticada, el teorema CAP ha sido ampliamente adoptado por grandes compañías así como por la comunidad NoSQL.

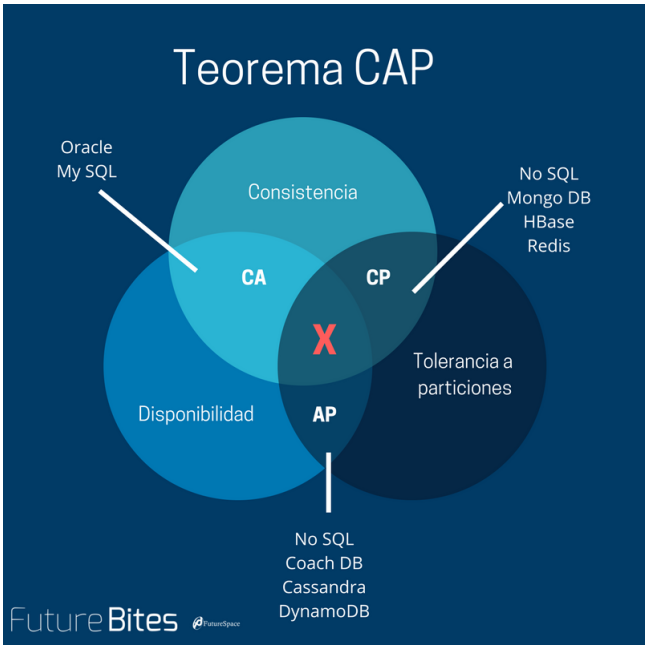


Fig. 4.2. Teorema CAP [37]

En este teorema se establece la idea de que en un entorno distribuido con datos compartidos no se pueden mantener de forma continua las la consistencia perfecta, la dispo-

nibilidad y la tolerancia a fallos simultáneamente. La importancia de este teorema reside en entender que para ganar velocidad debe sacrificarse por lo menos una de las características mencionadas anteriormente, y que por lo tanto se deben hacer combinaciones de esas características teniendo en cuenta las siguientes posibilidades [38]:

- CP: sistemas consistentes y tolerantes a fallos. El sistema podrá ejecutar las operaciones de forma consistente aunque se pierda la comunicación entre los nodos de la red. No obstante, no se asegura la disponibilidad de los datos.
- AP: sistemas disponibles y tolerantes a fallos. El sistema estará siempre disponible para responder a las peticiones que le lleguen, a pesar de que se pierda la comunicación entre los nodos. Sin embargo, los datos pueden qudar inconsistentes.
- CA: sistemas consistentes y disponibles. El sistema responderá continuamente a las peticiones que le lleguen y los datos siempre serán consistentes. Sin embargo, con estas características, no se permite una pérdida de comunicación en la red..

Es importante conocer estas características para tener en cuenta las ventajas e inconvenientes de cada tipo de bases de datos. Brewer indica que se debe utilizar como criterio de selección las características que se consideren más críticas para el negocio [39].

Además de ello, se debe tener también en cuenta las denominadas propiedades ACID, que se implementan en las bases de datos tradicionales y las propiedades BASE que se implementan en las bases de datos NoSQL, ya que las características de dichas propiedades en conjunto con el teorema CAP enunciado por Brewer será lo que determine la decisión final en base a las necesidades del negocio. Previamente a que realicemos este paso en el proyecto que nos ocupa debemos mencionar qué son las propiedades ACID y BASE para tener una visión completa de las propiedades que caracterizan a cada modelo.

4.2.1. Propiedades ACID

Como norma general, toda persona o empresa que necesita almacenar datos quiere que su sistema sea lo más fiable posible y que ante cualquier imprevisto los datos no sufran ninguna modificación. En el caso de un banco, esta situación puede ser especialmente crítica, ya que ante los movimientos de dinero en las cuentas el negocio necesita que no se pierda ninguna operación. Es en este momento cuando surge el concepto de transacción en bases de datos; una transacción se define como la unidad de trabajo de un sistema, normalmente estará compuesto por varias operaciones y asegura que se ejecuten todas o no se ejecute ninguna de ellas, evitando así que el sistema se quede en un estado inconsistente.

Para que una transacción sea considerada como tal debe cumplir con las cuatro propiedades denominadas ACID [40]:

- Atomicidad: es la propiedad que hace referencia a la realización de una operación o no. Así se asegura que dado un conjunto de operaciones se ejecutan todas o no se ejecuta ninguna, por lo que en caso de ocurrir algún fallo el sistema no queda a medias.
- Consistencia: mediante esta propiedad se conserva la consistencia de la base de datos, ya que únicamente se ejecutan aquellas operaciones que no violan las reglas de integridad establecidas.
- Aislamiento: esta propiedad conserva la independencia de las operaciones. Esta propiedad asegura que en caso de realizar varias transacciones sobre la misma información se ejecutarán de manera aislada, sin generar ningún tipo de error. Mediante esta propiedad también se puede ajustar el grado de aislamiento entre las transacciones
- Durabilidad: esta propiedad asegura que, si una transacción se realiza correctamente, el sistema garantiza que ésta persistirá aunque el sistema falle.

Para que un conjunto de operaciones sea tratado como una transacción se debe marcar el conjunto mediante una directiva y una vez finalizado confirmar la operación si todo fue bien (commit) o en caso de haber algún problema, deshacer todas las operaciones devolver la base de datos al estado consistente anterior (rollback).

Una transacción pueda denominarse "fiableçuando cumple estas cuatro propiedades ACID, y resulta un elemento esencial a la hora de preservar la consistencia de las operaciones en un sistema de base de datos.

4.2.2. Propiedades BASE

Como se ha mencionado en el punto anterior, ACID es una característica muy importante en las bases de datos tradicionales; sin embargo, en los sistemas NoSQL se optó por implantar un ACID “relajado” buscando conseguir una gran escalabilidad. Así, mediante esta relajación de ACID se redefinieron algunas cualidades de este modelo, concretamente la consistencia y la durabilidad para así favorecer la disponibilidad y el rendimiento.

Esta relajación de las propiedades ACID se denominó BASE [41] y hace referencia a tres conceptos:

- Basic Availability: el sistema funciona la mayor parte del tiempo.
- Soft-State: el sistema no tiene por qué ser consistente con las operaciones de escritura, al igual que las diferentes réplicas tampoco tienen por qué ser consistentes todo el tiempo.
- Eventual Consistency: en el sistema la consistencia se da únicamente en momentos eventuales.

El sistema BASE es opuesta a ACID ya que éste es pesimista [35] y fuerza la consistencia al final de cada operación, mientras que BASE es optimista y permite que la consistencia de los datos se dé de manera puntual. Por otro lado, la disponibilidad en BASE se logra a través de mecanismos de soporte a fallos parciales [17], consiguiendo así que el sistema esté siempre funcionando. De esta manera, se consigue alcanzar niveles de escalabilidad que sería imposible conseguir con las propiedades ACID.

4.2.3. Selección de los sistemas de bases de datos a comparar

Una vez expuestas las propiedades mencionadas en el apartado anterior, nos vemos con la capacitación suficiente para escoger los paradigmas y sistemas de bases de datos a comparar.

4.2.3.1. Selección de paradigmas

En este punto se va a seleccionar uno de los modelos dentro de las bases de datos tradicionales y otro dentro de las bases de datos NoSQL, atendiendo a las necesidades que resulten más importantes para realizar este proyecto.

Comenzando con el teorema CAP y como se ha mencionado, tanto los sistemas de bases de datos NoSQL como los tradicionales están restringidos a cumplir únicamente dos de las tres propiedades que lo conforman. No obstante, este teorema se enuncia en un entorno en el que los sistemas (independientemente de que se trate de uno tradicional o un NoSQL) trabajan de forma distribuida y con datos compartidos. Esta premisa se aleja del objetivo de este proyecto pues simplemente queremos comprobar la eficiencia de un sistema tradicional comparado con un sistema NoSQL. Por tanto no dispondremos de un entorno distribuido ni de datos compartidos ya que únicamente poseemos una máquina en la que desarrollar el proyecto, tal y como se especificará más adelante.

Si analizamos cada una de las posibilidades del teorema CAP nos encontramos con que los sistemas que obedecen a las características C-A no son otros que los sistemas de datos tradicionales como por ejemplo los modelos relacionales. Dicho esto, esta característica será de obligado cumplimiento dada la naturaleza del proyecto de querer comparar el rendimiento entre un modelo tradicional y uno NoSQL. En cuanto a las dos propiedades restantes del teorema CAP, ambas implementan la premisa de trabajar de forma distribuida, que como ya hemos dicho no se aplica a este proyecto. No obstante, dado que hemos escogido una de las propiedades que implementa la consistencia, es de esperar que para establecer una comparación con el modelo tradicional también se encuentre esta propiedad en el modelo NoSQL. Por tanto, nos hemos decidido por un sistema relacional que implementa Consistencia y Disponibilidad y un NoSQL que implementa Consistencia y Tolerancia a particiones.

Como es de esperar, el modelo relacional escogido implementa las propiedades ACID explicadas anteriormente. Sin embargo, en el sistema NoSQL aun debemos escoger el

paradigma de almacenamiento de datos que cumpla la característica de Consistencia y Tolerancia a particiones, para lo cual utilizamos las propiedades BASE. Uno de los paradigmas que cumple esta serie de propiedades es el orientado a documento, y dado el conocimiento previo que se poseía con este tipo de almacenamiento y la relevancia que está adquiriendo en los últimos tiempos, debido a su gran flexibilidad y velocidad, ha sido el que se ha escogido. También han influido en esta decisión la simplicidad de la estructura documental a través de la notación en clave-valor.

Finalmente, mencionar que se ha elegido el paradigma orientado a documentos por encima del columnar, por ejemplo, ya que este es más indicado para cuando hay una cantidad de datos muy grande y se quieren aplicar técnicas de Big Data o Business Intelligence sobre los datos, y este no era el objetivo del trabajo. Por otro lado, una base de datos orientada a grafos tampoco era una buena elección para el propósito que nos ocupa ya que estas bases de datos están pensadas para estructuras en las que se les da mucha importancia a las relaciones entre los nodos de datos que forman el grafo, como es el caso de las redes sociales, foros o similares.

4.2.3.2. Selección de software

Una vez elegidos los paradigmas que se van a utilizar, se debían elegir los sistemas de bases de datos concretos que se van a comparar, es decir, el software específico que se iba a utilizar para cada uno de ellos. Dentro de los sistemas relacionales, dado su antigüedad y su enorme utilización hay multitud de soluciones diferentes que implementan este paradigma. Algunos de los sistemas más conocidos son Oracle, MySQL o SQL Server. De estos mencionados, se ha optado por elegir el sistema gestor Oracle dada su gran popularidad en el mundo empresarial y porque el desarrollador ya tenía conocimientos de este sistema.

Por otro lado, en dentro de los sistemas NoSQL orientados a documentos, los sistemas son más escasos, ya que este tipo de bases de datos son relativamente nuevos y menos conocidos. De todos modos, últimamente están adquiriendo una gran popularidad y cada vez son más utilizadas en entornos empresariales. Algunos ejemplos de bases de datos documentales son MongoDB o CouchDB.

Concretamente se ha optado por elegir el sistema MongoDB debido al auge que este sistema ha experimentado en los últimos años. Otra de las razones por las que se ha elegido ha sido porque el desarrollador poseía conocimientos previos de MongoDB y de Javascript, pues la consola de MongoDB funciona como intérprete de este lenguaje y a través del mismo permite realizar operaciones sobre los datos.

Por último, cabe destacar que aunque se han elegido estos dos sistemas para el presente trabajo, se podrían haber elegido otros sistemas cualesquiera, así como otros paradigmas de almacenamiento. Se ha optado por los ya mencionados debido a que el estudio del rendimiento sobre estos sistemas pretende ayudar a resolver la incógnita sobre cuál

utilizar, dada su popularidad y uso extendido por parte de las empresas. Por tanto el estudio del rendimiento de estos sistemas puede desembocar de manera más adecuada en el objetivo que se pretende conseguir: poner de manifiesto la importancia de elegir.

4.3. Catálogo de requisitos

En este punto se establecen los requisitos que afectan al desempeño del proyecto tras realizar un análisis de los objetivos que se pretendían conseguir. El catálogo de requisitos aquí expuesto nos servirá para llevar a cabo la completitud del trabajo que nos ocupa.

Para la organización de estos requisitos nos hemos decidido por un diseño de tabla en el que se recogerá la información de cada uno de ellos. El modelo de tabla utilizado es el siguiente:

Identificador	RP-XX
Título	-Título del requisito-
Descripción	-Descripción del requisito-
Prioridad	Alta-Media-Baja

Tabla 4.1. PLANTILLA DE REQUISITOS

Donde los campos de la tabla anterior se explican a continuación:

- Identificador: cada requisito tendrá un identificador único que permitirá diferenciarlos a lo largo del ciclo de vida del proyecto. El identificador utilizado tendrá la forma RP-XX donde RP hace referencia a "Requisito del Proyecto" y XX a un número incremental.
- Título: título de cada requisito.
- Descripción: información detallada del requisito.
- Prioridad: hace referencia a la importancia que se le da a cada requisito. Esta podrá ser alta, media o baja.

El catálogo de requisitos establecidos es el siguiente:

Identificador	RP-01
Título	Búsqueda del conjunto de datos
Descripción	Realizar la búsqueda de un conjunto de datos con datos estructurados que se puedan modelar fácilmente tanto en bases de datos relacionales como en MongoDB en formato json, csv o tsv que pueda ser importado en MongoDB
Prioridad	Alta

Tabla 4.2. RP-01: BÚSQUEDA DEL CONJUNTO DE DATOS

Identificador	RP-02
Título	Esquema conceptual en MongoDB
Descripción	Diseñar el esquema conceptual de los documentos que compondrán la colección en MongoDB
Prioridad	Alta

Tabla 4.3. RP-02: ESQUEMA CONCEPTUAL EN MONGODB

Identificador	RP-03
Título	Esquema conceptual Oracle
Descripción	Diseñar el esquema conceptual de las tablas que compondrán la base de datos relacional
Prioridad	Alta

Tabla 4.4. RP-03: ESQUEMA CONCEPTUAL DE LA BASE DE DATOS RELACIONAL

Identificador	RP-04
Título	Modelo relacional
Descripción	Diseñar el modelo relacional a partir del esquema conceptual hecho para la base de datos relacional
Prioridad	Alta

Tabla 4.5. RP-04: MODELO RELACIONAL

Identificador	RP-05
Título	Transformación de los datos para MongoDB
Descripción	Transformar los datos iniciales para insertarlos en MongoDB
Prioridad	Alta

Tabla 4.6. RP-05: TRANSFORMACIÓN DE LOS DATOS PARA MONGODB

Identificador	RP-06
Título	Transformación de los datos para Oracle
Descripción	Transformar los datos iniciales para adecuarlos al modelo relacional diseñado
Prioridad	Alta

Tabla 4.7. RP-06: TRANSFORMACIÓN DE LOS DATOS PARA ORACLE

Identificador	RP-07
Título	Creación de la base de datos e inserción de los datos en MongoDB
Descripción	Crear la estructura de la base de datos e insertar los daos en MongoDB
Prioridad	Alta

Tabla 4.8. RP-07: CREACIÓN DE LA BASE DE DATOS E INSERCIÓN DE LOS DATOS EN MONGODB

Identificador	RP-08
Título	Creación de la base de datos e inserción de los datos en Oracle
Descripción	Crear la estructura de la base de datos e insertar los daos en Oracle
Prioridad	Alta

Tabla 4.9. RP-08: CREACIÓN DE LA BASE DE DATOS E INSERCIÓN DE LOS DATOS EN ORACLE

Identificador	RP-09
Título	Diseño de las consultas a realizar
Descripción	Diseñar las consultas que se van a realizar para llevar a cabo la comparativa
Prioridad	Alta

Tabla 4.10. RP-09: DISEÑO DE LAS CONSULTAS A REALIZAR

Identificador	RP-10
Título	Implementación de las consultas en Javascript
Descripción	Implementar las consultas en el lenguaje Javascript para ejecutarlas en MongoDB
Prioridad	Alta

Tabla 4.11. RP-10: IMPLEMENTACIÓN DE LAS CONSULTAS EN JAVASCRIPT

Identificador	RP-11
Título	Implementar las consultas en SQL
Descripción	Implementación de las consultas en el lenguaje SQL para ejecutarlas en Oracle
Prioridad	Alta

Tabla 4.12. RP-11: IMPLEMENTACIÓN DE LAS CONSULTAS EN SQL

Identificador	RP-12
Título	Ejecución de consultas
Descripción	Ejecución de las consultas implementadas en sus respectivos entornos Oracle y MongoDB
Prioridad	Alta

Tabla 4.13. RP-12: EJECUCIÓN DE CONSULTAS

Identificador	RP-13
Título	Análisis de los resultados
Descripción	Realizar un análisis de los resultados obtenidos tras la ejecución de las pruebas diseñadas
Prioridad	Alta

Tabla 4.14. RP-13: ANÁLISIS DE LOS RESULTADOS

5. DISEÑO

Tras realizar el análisis y obtener los requisitos, en este apartado se contará en detalle el proceso seguido para, a partir de los datos conseguidos, realizar los esquemas conceptuales del almacén de datos NoSQL, así como de la base de datos relacional. Asimismo también se detallan las consultas que se han elegido para realizar el estudio, así como una explicación de por qué se han elegido ese tipo de consultas.

5.1. Dominio y datos

Para realizar la comparativa de rendimiento entre un sistema de base de datos relacional y un sistema NoSQL se ha buscado un dominio y un conjunto de datos lo suficientemente grande como para que pudiera dar juego a realizar consultas complejas y que requieran un cierto estudio de los datos para poder trabajar mejor con ellos (requisito RP-01: Búsqueda del conjunto de datos).

Resulta importante destacar que una condición necesaria de la información contenida en el conjunto de datos (en adelante se utilizará la voz inglesa *dataset*) era que debía disponer de una estructura claramente definida. En el caso que nos ocupa, con MongoDB, tenemos que los datos pueden presentarse de una manera muy caótica, es decir, los documentos de una colección pueden tener una estructura dinámica y cada documento puede ser completamente diferente. Aplicando esta situación a nuestro problema en que debíamos trasladar la información del *dataset*) de MongoDB a un diseño relacional, de haberse dado esta casuística, resultaría prácticamente imposible adaptar la información a un modelo relacional. Es por esto que tal y como se ha dicho las colecciones almacenan documentos en los que únicamente varía la información propiamente dicha; manteniendo la estructura del documento y los atributos.

Por lo tanto, se ha optado por buscar un *dataset*) que cumpliese estos requisitos de estructura y con un formato que fuera fácilmente importable a MongoDB y que tuviera al menos entre dos y tres Gigabytes en datos.

Finalmente se ha escogido un *dataset* en formato .tsv (documentos que representan datos en forma de tabla en la que las columnas se separan por tabulaciones y las filas por saltos de línea) con temática cinematográfica que se puede encontrar y descargar libremente desde la página web de IMDB[13].

Aunque el *dataset* completo está formado por un total de siete colecciones, para el presente proyecto únicamente se han utilizado cuatro de ellas, dadas sus características en cuanto a variedad de tipos de datos y su contenido en cuanto a posibilidades de consultas, es decir, las colecciones restantes no aportaban dinamismo a la hora de realizar consultas

[13] https://www.imdb.com/interfaces/

que supusieran una carga de trabajo para la base de datos. Por otro lado, dadas las limitaciones de espacio, se ha prescindido de su utilización. A continuación se explica en qué consiste cada colección y se explican algunas de sus características:

- Title Akas: contiene información acerca de los títulos de las películas como si el título es el original, en qué idiomas está o en qué región se hizo.
- Title Basics: contiene información acerca de las películas y series como su género, año de comienzo y de fin o su duración.
- Title Crew: contiene la información relativa a los directores y escritores de todas las películas y series almacenadas.
- Name Basics: contiene toda la información relativa a los actores; por qué películas son famosos, en qué año nacieron/fallecieron o qué profesiones tienen.

Inicialmente también se pensó en añadir una quinta colección llamada Principals, la cual contenía información acerca de todo el reparto que había participado en las películas. No obstante, al descargar esta colección, la información contenida no coincidía con la indicada en la documentación del dataset. Somos conscientes de que la información relativa a esta colección es la utilizada para establecer una relación entre la colección que almacena las personas y la que almacena las películas, por tanto, sería importante que se incluyera. Esta opción se ha descartado por el inconveniente mencionado de que desconocíamos la estructura del dataset por falta de documentación y porque dado el gran tamaño del archivo que se descargaba (cercano a los 3GB), resultaba muy complicado trabajar con ella en el equipo del que se disponía. No obstante, hemos sido capaces de establecer relaciones entre una persona y una película a través de la información relativa a los directores/escritores almacenada en la colección Title Crew y las películas por las que las personas son conocidas, cuya información se obtiene de la colección Name Basics.

5.2. Diseño de la base de datos NoSQL

Como ya se ha indicado anteriormente, una de las principales características de las bases de datos NoSQL es que no tienen por qué tener una estructura fija, sino que puede ser variable. En el caso de MongoDB, los documentos de las colecciones tampoco tienen por qué tener una estructura fija y cada documento puede tener un número variable de campos y de distintos tipos. Teniendo esto en cuenta, para realizar la base de datos en MongoDB, no se ha realizado un diseño como tal debido a que, por un lado, dada la naturaleza de este tipo de bases de datos no resulta necesario, y por otro lado porque al descargar los archivos tsv que incorporaban los datos, éstos ya poseen una estructura predefinida (requisito RP-02: Esquema conceptual en MongoDB). Esta estructura queda reflejada en la siguiente imagen:

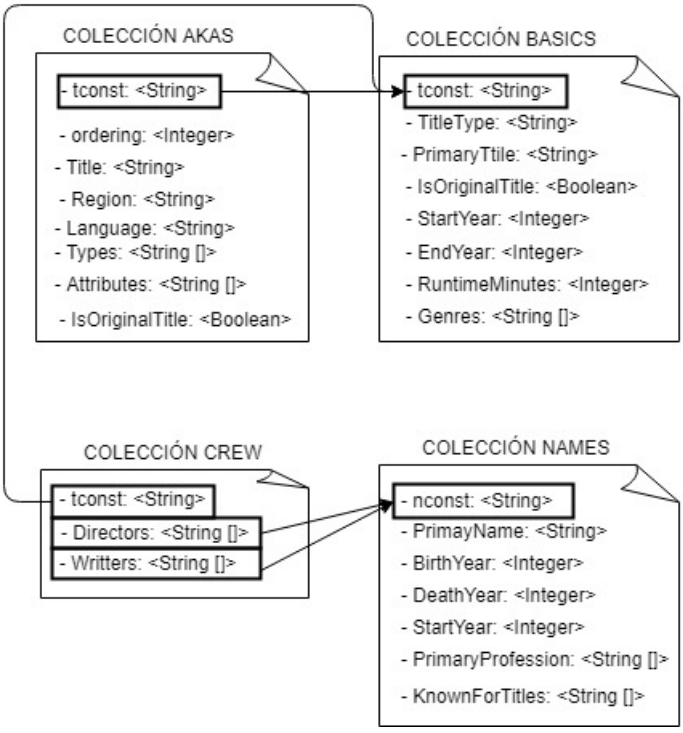


Fig. 5.1. Diagrama estructura en colecciones

Como se puede observar, la base de datos utilizada consta de cuatro colecciones distintas que contienen información relativa a los actores, a las películas y a los directores/escritores de las mismas. Aunque en las bases de datos NoSQL no existe como tal el concepto de clave ajena, se puede observar que en todas las colecciones se repite el campo “tconst”, el cual representa un identificador para cada película. De igual manera, el campo “nconst” representa un identificador unívoco para cada persona. Por tanto podemos establecer “relaciones” entre las colecciones gracias a estos campos, y que quedan reflejadas en la imagen mediante la unión de las flechas.

5.3. Diseño de la base de datos relacional

Para el diseño se ha tenido en cuenta que se ha escogido el paradigma relacional dentro de las bases de datos tradicionales, y concretamente el Sistema Gestor de Bases de Datos (SGBD) Oracle. Éste, como es obvio, implementa el paradigma relacional, por lo que las bases de datos que se creen en este SGBD deben seguir este modelo. Dado que el conjunto de datos inicialmente descargado era una base de datos NoSQL orientada a documentos, a pesar de tener cierta estructura, no se puede aplicar directamente sobre una base de datos relacional. Por lo que ha sido necesario realizar un análisis de los datos obtenidos para realizar un diseño conceptual (requisito RP-03: Esquema conceptual de la base de datos relacional) de la base de datos relacional lo más eficiente posible.

La finalidad de realizar en primer lugar un diseño conceptual es conseguir una mayor abstracción mediante la representación de objetos del mundo real como entidades y relaciones entre dichas entidades [42].

Así, tras realizar el análisis, se ha realizado el siguiente diagrama conceptual:

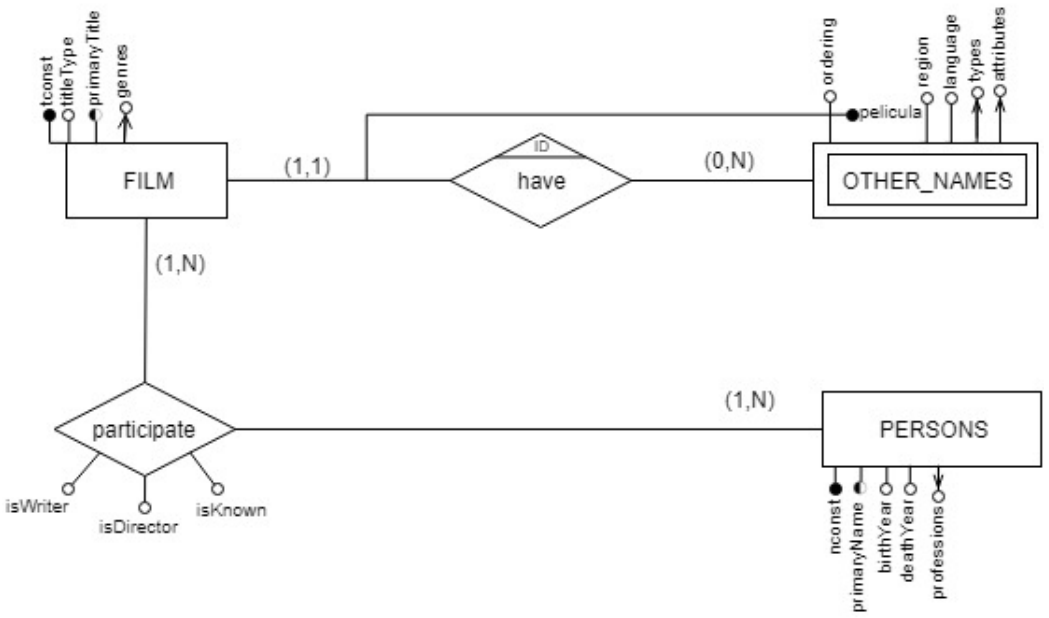


Fig. 5.2. Diagrama E/R

A continuación, se detallan los supuestos semánticos aplicados a este diagrama:

- Toda película debe tener al menos un director.
- Toda película debe tener al menos un escritor o guionista.
- Toda persona almacenada debe ser conocida por al menos una película.
- FILM.genres en {“Drama”,“Action”, “Documentary”, “Short”, “Animation”, “News”, “Romance”, “Comedy”}.

- OTHER_NAMES.types en {“alternative”, “dvd”, “festival”, “tv”, “video”, “working”, “original”, “imdbDisplay”}.
- OTHER_NAMES.attribute en {“InformalTitle”, “PremiereTitle”, “CompleteTitle”, “PosterTitle”, “ShortTitle”, “PromotionalTitle”}.
- PERSONS.professions en {“Actor”, “Actress”, “Director”, “Writer”, “Photographer”,“Miscelaneous”, “Producer”, “Composer”, “Soundtrack”}.

Se ha realizado este diagrama buscando un diseño eficiente, intentando evitar en todo lo posible los valores nulos así como almacenar información redundante; aunque se podría haber optado por otros diseños.

Una posible alternativa podría ser fusionar las entidades “OTHER_NAMES” y “FILM” en una única entidad, ya que ambas contienen información relevante a las películas. No obstante, esta alternativa se ha descartado debido a la naturaleza de los datos almacenados; la entidad “OTHER_NAMES” almacena el nombre que recibe una misma película en distintos países, así como información relativa al tipo de película que es o el idioma, mientras que en la entidad “FILM” se almacena información más general de la película como el título original o el género. Si se hubiesen fusionado ambas entidades se repetiría toda la información de la entidad “FILM” para cada entrada de la entidad “OTHER_NAMES”, lo cual no es eficiente ya que la misma información estaría duplicada a lo largo de la tabla en la base de datos y se debería controlar la consistencia entre las distintas filas. Con el diseño realizado se consigue evitar este problema separando la información de la película original de las distintas versiones que se han creado para cada país.

Por otro lado, analizando los datos, también se podría pensar en modelar los atributos “directores” y “escritores” de la entidad “FILM” como una especificación de la entidad “PERSONS”, en el cual se definiría una superclase persona y las subclases serían director y escritor. Esta alternativa no se ha valorado ya que en los datos originales no se hacía ninguna distinción entre actores, directores y escritores, sino que todos estaban almacenados en la misma colección. Además, estas entidades tampoco poseen relaciones por sí solas, sino que únicamente aparecen como atributos de la entidad “FILM” Por último, una misma persona puede tener varios roles en la misma película, por ejemplo, siendo director, actor y escritor de la misma.

Como se ha comentado en el punto 5.1 Dominio y datos , no hemos sido capaces de utilizar la colección Principals para la realización de este proyecto. Esto es un inconveniente puesto que es esta colección la que establece la relación entre películas y personas necesaria para poder utilizar los datos orientado al modelo relacional. Es debido a esto que se ha tomado la decisión de modelar la relación “participate” a través de los atributos “isDirector”, “isWriter” y “isKnown”, dando lugar a una relación de participación en la que como es obvio, no tendremos contemplada una relación que cubra absolutamente todos los registros de actores para películas, puesto que se puede dar el caso de que un actor haya participado en una película pero no sea ni reconocido por ella ni escritor ni

director de la misma. Precisamente esta, era la información que quedaba reflejada por la colección Principals, pero como ya se ha dicho no hemos podido utilizar dicha colección por lo que tuvimos que aplicar esta alternativa. Mediante estos atributos (“isDirector”, “isWriter” y “isKnown”), se representan estas características de las personas, que en los datos originales estaban modelados como un array. Dada la característica de que en el modelo relacional cada registro debe contener un único valor, se ha optado por estos atributos binarios en los que se indicará con un 1 si la persona es alguna de las tres cosas y con 0 si no lo es. Una posible alternativa hubiera sido modelar estas relaciones como tablas independientes, creando una tabla para cada atributo. No obstante, estas tablas únicamente estarían compuestas por el identificador de la persona y el identificador de la película, y serían las tres tablas prácticamente idénticas. Con el diseño realizado se evita tener estas tablas repetidas con la misma estructura y se modela todo como una única relación.

Finalmente, a partir de este diagrama E/R hemos realizado las conversiones pertinentes y obtenido el diagrama relacional (requisito RP-04: Modelo relacional) final a partir del cual hacer la implementación de la base de datos. Este diseño relacional final se muestra en la siguiente imagen:

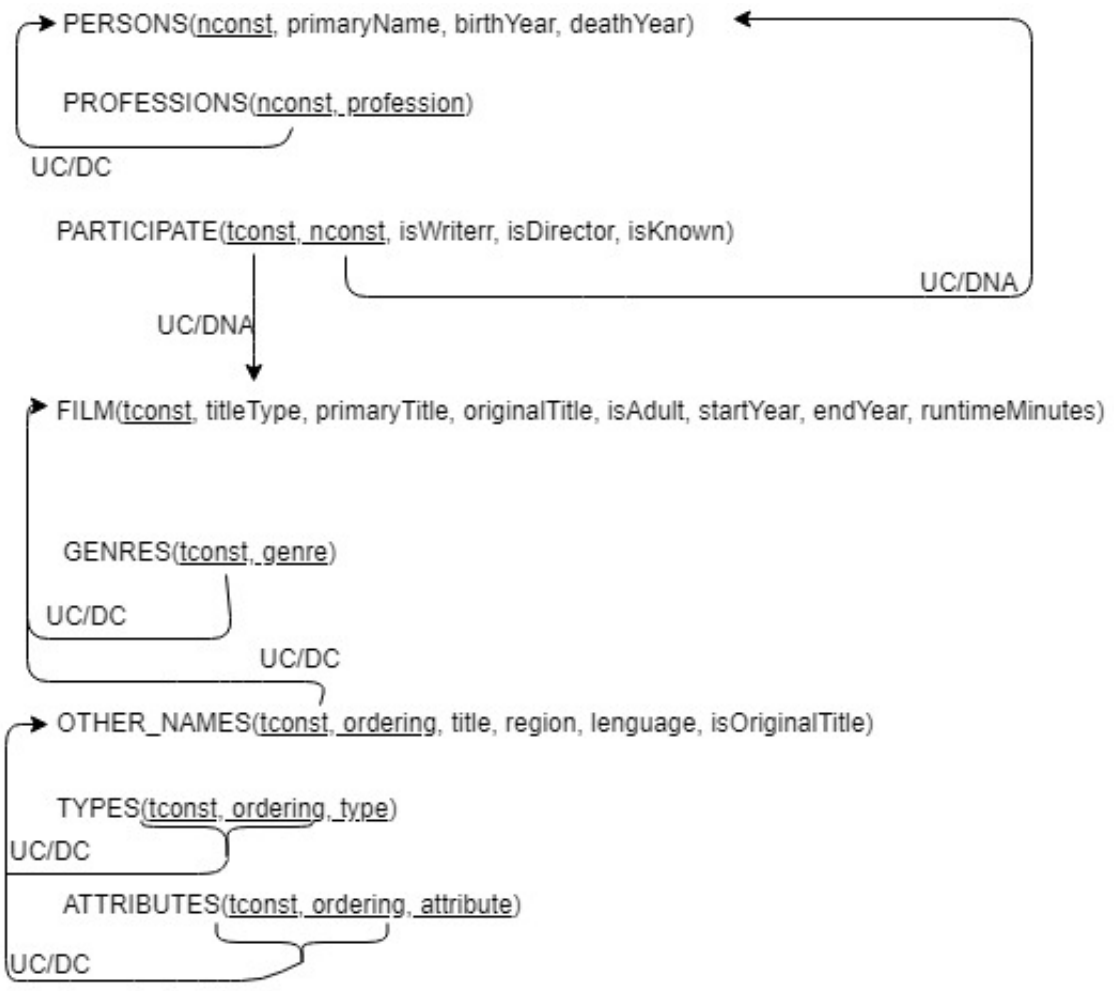


Fig. 5.3. Diagrama Relacional

En este caso, debido a la pérdida de semántica sufrida en la transformación, los supuestos semánticos presentados en el diseño relacional deberán ser transformados en estructuras propias del sistema gestor Oracle:

- Toda película debe tener al menos un director. → Se transformará en una aserción sobre las tabla FILM y PERSONS para comprobar que toda película tiene algún director.
- Toda película debe tener al menos un escritor o guionista. → Se transformará en una aserción sobre las tabla FILM y PERSONS para comprobar que toda película tiene algún escritor o guionista.
- Toda persona almacenada debe ser conocida por al menos una película. → Se transformará en una aserción sobre las tabla FILM y PERSONS para comprobar que toda persona es conocida por alguna película.
- FILM.genres en {“Drama”,“Action”, “Documentary”, “Short”, “Animation”, “News”, “Romance”, “Comedy”}. Se transformará en un CHECK sobre la tabla FILM.
- OTHER_NAMES.types en {“alternative”, “dvd”, “festival”, “tv”, “video”, “working”, “original”, “imdbDisplay”}. Se transformará en un CHECK sobre la tabla OTHER_NAMES.
- OTHER_NAMES.attribute en {“InformalTitle”, “PremiereTitle”, “CompleteTitle”, “PosterTitle”, “ShortTitle”, “PromotionalTitle”}. Se transformará en un CHECK sobre la tabla OTHER_NAMES.
- PERSONS.professions en {“Actor”, “Actress”, “Director”, “Writer”, “Photographer”,“Miscelaneous”, “Producer”, “Composer”, “Soundtrack”}. Se transformará en un CHECK sobre la tabla PERSONS.

Para realizar la conversión entre el diagrama E/R original y este modelo relacional se han utilizado las reglas de transformación de E/R a relacional vistas durante la carrera [43]. Así, mediante estas reglas, los atributos multivaluados como géneros, atributos o profesiones se han convertido en nuevas tablas que tienen como clave primaria la clave primaria de la tabla original junto a un valor de dominio del atributo multivaluado. Por otro lado, la relacione N:M “PARTICIPATE” se han transformado en una nueva tabla que tienen como clave primaria la concatenación de las claves primarias de las dos tablas que une. Por último, en la relación 1:N “HAVE” se ha hecho la propagación de clave de la tabla con cardinalidad 1 a la tabla con cardinalidad N, dando como clave primaria de la tabla “OTHER_NAMES” la concatenación de ambas claves primarias.

Finalmente destacar que se ha incluido como regla de integridad referencial el borrado y actualización en cascada para los atributos multivaluados y las relaciones de identificación aunque esto se contemplaba en la base de datos original de MongoDB. Asimismo,

en las relaciones N:M se realizarán las actualizaciones en cascada pero los borrados no se propagarán.

5.4. Diseño de Consultas

En este punto se explican cuáles han sido las consultas que se han elegido para realizar la comparativa entre los dos sistemas de almacenamiento (requisito RP-09: Diseño de las consultas a realizar). Se han elegido consultas que impliquen trabajar con una gran cantidad de elementos y que puedan requerir una gran cantidad de operaciones por parte del ordenador, como pueden ser operaciones de combinación de datos, ordenaciones, agrupaciones...

Se ha elegido este tipo de consultas debido a que son operaciones relativamente pesadas para una CPU así como por el hecho de que la base de datos NoSQL MongoDB originalmente no estaba pensada para realizar algunos tipos de operaciones, como pueden ser las combinaciones entre datos de distintas colecciones. Esta característica fue añadida posteriormente debido a la demanda de los usuarios.

Las consultas que se han elegido han sido las siguientes:

- En la colección Title Akas, contar el número de películas que se han realizado en un determinado país. Es una consulta que requiere una operación de agrupación, así como una operación matemática.
- En la colección Title Basic, obtener los distintos géneros que tiene una película. Esta consulta en MongoDB requiere consultar un array. Por otro lado, en Oracle esta consulta va a requerir realizar una combinación (join) y trabajar en memoria con un gran número de registros.
- En la colección Title Basic, obtener todas las películas que se han producido a partir de un determinado año y que cumplen una serie de características lógicas para finalmente ordenarlas por orden cronológico. Esta consulta requiere la utilización de operadores lógicos así como una ordenación en memoria de un número notable de elementos.
- En la tabla Name Basic, las edades de todas las personas que han nacido y muerto entre dos años concretos y realizar una ordenación por este nuevo valor calculado. Esta consulta requiere la realización de cálculos matemáticos así como de una ordenación en memoria.
- En la colección Title Crew, a partir de una película concreta y un director en concreto, obtener más información de esa película y ese director. Esta consulta implica realizar dos combinaciones (joins), con las colecciones Name Basic y Title Basic, y por lo tanto, trabajar en memoria un gran número de registros.

En el siguiente capítulo se procederá a codificar estas consultas en el lenguaje de cada Software escogido para, de esta manera, poder ejecutarlas y estudiar el rendimiento obtenido con cada una de ellas.

6. IMPLEMENTACIÓN

Como ya se ha comentado, en el presente TFG se realiza una comparativa de rendimiento entre una base de datos relacional y un almacén de datos NoSQL. Ya se han visto las distintas alternativas que existen actualmente en el mercado y lo que ofrece cada una de ellas, por lo que debíamos elegir dos sistemas para realizar la comparativa. A partir de ese punto, y una vez determinado el diseño que se va a seguir para realizar dicha comparativa, se debe ejecutar el plan de acción propuesto (las consultas especificadas en el capítulo anterior) para lo cual primeramente debemos construir el modelo sobre el que se trabajará.

Para ello en las líneas siguientes explicamos como se ha realizado el proceso de construcción del entorno de trabajo para los dos gestores elegidos: MongoDB en el caso del sistema NoSQL y Oracle en el caso del sistema relacional.

6.1. MongoDB

MongoDB es un sistema de base de datos NoSQL que implementa el paradigma basado en documentos. Aunque una de las principales ventajas de este sistema es su escalabilidad, también destaca por su potencia y su flexibilidad. En las bases de datos orientadas a documentos se sustituye el concepto de fila, típico de los sistemas relacionales, por un modelo más flexible como son los documentos [44].

En MongoDB la información se almacena como documentos BSON (Binary Javascript Object Notation), una representación binaria de los documentos JSON y que por tanto, ocupan menos espacio. Los documentos están compuestos de varios campos clave-valor y en general tienen la siguiente estructura:

Fig. 6.1. Estructura general de un documento [45]

Así, tenemos que los campos que forman los documentos pueden ser de cualquier tipo incluido en BSON, o incluso arrays, otros documentos o arrays de documentos.

En este sistema de almacenamiento la terminología cambia un poco respecto a una base de datos relacional. En ambos sistemas existe el concepto de base de datos y tiene el mismo significado; sin embargo, una tabla del sistema relacional en MongoDB sería el

equivalente a una colección. Por otro lado, una fila de una tabla del relacional en MongoDB recibiría el nombre de documento.

6.1.1. Tratamiento de los datos

Dado que al descargar los datos, éstos presentaban formato tsv, la inserción de los datos en MongoDB prácticamente no requirió ningún tipo de tratamiento. Este formato de datos puede ser importado directamente en MongoDB sin ningún problema, al igual que los formatos json o csv.

El único tratamiento (requisito RP-05: Transformación de los datos para MongoDB) destacable que hubo que realizar fue debido al hecho de que al producirse la importación en de las colecciones de MongoDB, los campos que debían ser arrays no se importaban como tal, sino como un dato tipo String separado por comas. Esto era debido a que, al disponer únicamente de los fichero con formato tsv, es necesario utilizar el comando `mongoimport` para realizar la inserción de los datos, y en estos ficheros no se guarda información relativa al tipo de datos que era cada atributo originalmente. Esta situación habría sido distinta de disponer directamente los datos en formato BSON, ya que en éstos sí se mantienen exactamente los tipos e información de los datos almacenados en MongoDB. Puesto que esto no resulta ser así, la importación de cualquier otro formato distinto a éste (como es el caso que nos ocupa) produce este tipo de inconvenientes.

Por otro lado, los campos que eran nulos venían representados con el string "\\N" y se decidió sustituir estos valores por el valor "NULL" para que representase la información de manera correcta con la sintaxis que utiliza MongoDB.

Para realizar estos cambios en los datos una vez que ya habían sido insertados, se ha aprovechado la característica que posee MongoDB de interpretar código JavaScript y se ha creado un script en este lenguaje de programación que realiza la actualización de los documentos con los cambios que se han mencionado. En el caso de la sustitución de los valores representados con los caracteres "\\N" por el valor "NULL" simplemente se inspecciona cada colección buscando dichos valores en sus atributos, y si el resultado es coincidente se actualiza el documento para que en lugar de este símbolo tenga el valor "NULL". Para el caso de la actualización de los strings que debían ser arrays, se procede a crear un algoritmo que seleccione los atributos que se pretenden convertir (ya que se conocía cuales eran gracias a la documentación que proveía el dataset), separe el string por las comas e introduzca en un array cada uno de los elementos obtenidos al realizar la separación. Posteriormente a esto se actualiza el campo que antes contenía el string para que contuviese el array construido.

Cabe destacar que para ambas transformaciones de los datos, se tuvieron ciertas dificultades en cuanto a tiempo de ejecución. Éstas principalmente fueron debidas a que, dado el gran tamaño del conjunto de datos escogido, se debían actualizar un gran número de documentos. Para agilizar esta situación se decidió realizar las operaciones de actuali-

zación mediante el método "bulk" que permite MongoDB; es decir, se "precargan" todas las operaciones de actualización hasta llegar a un determinado número (dicho número se estableció en 500.000 documentos) y posteriormente se hacen persistentes todas esas operaciones. De esta forma, se consiguió que el proceso se realizara en menor tiempo gracias a que no se persistían los cambios por cada uno de los documentos que se actualizaban.

6.1.2. Creación del almacén de datos e inserción de datos

Como ya se ha mencionado anteriormente, los datos que se almacenan en un almacén de datos NoSQL no tiene por qué tener una estructura fija, sino que puede ser dinámica. Por esta razón no ha sido necesario realizar un estudio previo de los datos para definir claves primarias o alternativas. No obstante, a la hora de realizar las inserciones, MongoDB crea por defecto un campo adicional en cada fila insertada llamado "_id" de tipo ObjectID, el cual es un valor de 12 bytes que se utiliza para identificar cada fila de manera unívoca [46]. Aunque por defecto se cree de tipo ObjectID este campo puede ser modificado por el usuario para que se utilice otro campo como identificador de fila. En este caso, para realizar la comparativa, inicialmente se deja como "_id" el campo que genera el sistema automáticamente.

Finalmente, para poder insertar los datos, en primer lugar se ha creado un nuevo almacén de datos (requisito RP-07: Creación de la base de datos e inserción de los datos en MongoDB). Para hacerlo se ha utilizado el comando `use` nombre_BBDD, con el cual si ya existe una base de datos con ese nombre se sitúa en ella para poder comenzar a hacer operaciones, o si por el contrario, no existe ninguna base de datos con ese nombre crea una nueva instancia vacía en memoria para poder comenzar a trabajar en ella. Si tras un tiempo no se ha hecho ninguna operación y se cierra el servidor de MongoDB esa base de datos que se creó vacía se elimina automáticamente.

Una vez creada la base de datos, para realizar la inserción se utilizó el comando `mongoimport` . Fue necesario utilizar este comando ya que no se disponía de los archivos BSON con la información de la base de datos, sino que únicamente se disponía de los archivos en formato .tsv con los datos.

Para realizar una exportación de datos y que se guarde información relativa al tipo de dato de cada atributo MongoDB ofrece otro comando llamado `mongodump` con el cual, además de crear los ficheros con los datos se crean unos ficheros binarios adicionales en formato `BSON` (Binary Javascript Notation Object) que son los contienen toda información relativa a los atributos.

6.2. Oracle

Oracle Database es un Sistema Gestor de Bases de Datos Relacional (SGBDR).Este sistema, cuya primera versión se desarrolló en el año 1977, fue uno de los primeros sistemas en implementar el modelo relacional propuesto por Codd en esa misma década.

Como ya se ha explicado en el apartado Estado del Arte, este sistema relacional está basado en el concepto de tabla, las cuales están formadas por filas. Estas filas están compuestas por una serie de atributos que adquieren valores dentro de unos dominios. Los atributos a su vez reciben el nombre de columnas y no se permite que haya dos atributos con el mismo nombre. Cada fila es identificada de manera unívoca mediante un valor denominado clave primaria, la cual no puede ser repetida en distintas filas.

En el modelo relacional también se establece el concepto de relación, mediante el cual se permite establecer relaciones entre distintas tablas. Estas relaciones entre tablas se establecen a través de las denominadas claves ajenas, las cuales permiten identificar inequívocamente una fila en otra tabla ya que esta clave ajena se coloca en una tabla y hace referencia a la clave primaria de la tabla padre. Como es obvio, la fila referenciada por la clave ajena debe existir o sino se producirá un error en la consistencia de los datos. Las reglas mediante las cuales se asegura esta persistencia entre tablas recibe el nombre de restricciones de integridad.

Con el paso de los años, el software desarrollado inicialmente ha ido evolucionando y en cada nueva versión que salía al mercado se han ido añadiendo nuevas características; lo que ha llevado a este sistema a ser uno de los más conocidos del mundo. Pese a que su instalación y mantenimiento suele tener un alto coste, debido a que es un software propiedad de una empresa privada, no le ha impedido ser uno de los sistemas más utilizados en multitud de empresas de todo tipo.

6.2.1. Tratamiento de los datos

Ya que el gestor de Oracle implementa el modelo relacional de base de datos, en este caso sí que ha habido que hacer varios cambios respecto al diseño inicial de los datos (requisito RP-06: Transformación de los datos para Oracle). Ha sido necesario realizar un análisis de las estructuras definidas en las colecciones de MongoDB para realizar un diseño conceptual lo más eficiente posible; intentando evitar redundancias y gran cantidad de valores nulos.

Así, los principales cambios que se han realizado han sido los siguientes:

- En los archivos tsv con los datos originales, inicialmente los valores nulos eran representados por el string "\\N"pero para que el gestor de Oracle no los tratara como string más se han sustituido esos valores por “NULL”, que es la palabra reservada en Oracle para representar nulos.

- Los atributos que en la colección original se representaban como un array se han convertido en tablas N:M dado que según el modelo relacional cada celda únicamente puede contener un valor.

- Al hacer la conversión de los arrays a tablas N:M había muchos valores nulos, por ejemplo, había algunas películas que no tenían ningún género asociado y tenían el string "\\N". Al hacer esta conversión a tablas se ha optado por evitar introducir estos valores nulos en la nueva estructura, ya que no aportan ningún valor a los datos. En su lugar, se ha realizado la conversión únicamente de los valores que no eran nulos, dando por obvio que aquellos elementos que no se encuentren presentes en la nueva tabla es porque no tienen ningún valor asociado.

Para realizar estos cambios en los datos previamente de su inserción, se ha realizado un script en el lenguaje de programación Python mediante el cual se leían los archivos en formato tsv y se creaba un nuevo archivo con los datos necesarios. Se decidió utilizar este lenguaje debido a que permite trabajar con ficheros y con cadenas de texto de una manera más sencilla que otros lenguajes. Mediante este script se leía el archivo línea a línea y se iban aplicando los cambios necesarios para convertir los datos. En el caso de sustituir los valores nulos, se separaba cada elemento de la fila leída del fichero mediante el método `split()` con el cual se obtenía un array con los diferentes elementos que contenían cada líena. A continuación se recorría este array y si alguno de los elementos era igual al caracter "\\N" se sustituía por la palabra "NULL". De igual manera, para convertir los arrays en tablas N:M, se leía la fila completa y se buscaba la posición del identificador de la fina (tconst o nconst) así como la posición del array; se separaba este por sus elementos y se escribía en un nuevo fichero la información referente al identificador de la fila más el contenido de cada posición del array.

6.2.2. Creación de la base de datos e inserción de datos

A partir del diseño relacional descrito anteriormente, se ha creado un script en SQL para la creación de las tablas, así como para establecer las referencias entre las distintas tablas y las restricciones de integridad (requisito RP-08: Creación de la base de datos e inserción de los datos en Oracle).

Para facilitar la inserción de los datos, se decidió hacer la importación de los mismos mediante la herramienta SQL Developer, desde una opción que permite importar datos en diferentes formatos como tsv o JSON. Se tomó esta decisión porque, aunque en un principio sí que se creó el script de inserción de datos en SQL, este proceso tardaba mucho tiempo en construir la estructura para todas las filas y posteriormente el fichero creado ocupaba mucho espacio en disco.

Como problemas encontrados, destacar que durante la importación de los datos en las tablas, se sucedieron diversos problemas de violación de clave primaria debido a que se hacía referencia a una clave primaria de otra tabla que en realidad no existía. Esto era

debido a que, dada la estructura relacional necesaria en Oracle, era necesario establecer relaciones entre las distintas tablas para que la información fuera consistente. Para comprobar si esto era debido a un error cometido durante el tratamiento de los datos por parte del desarrollador o era una inconsistencia proveniente de los datos originales se hizo una consulta en la base de datos de MongoDB para confirmarlo ya que en este caso los datos se habían importado según se descargaron. Efectivamente se confirmó que era debido a un error de los datos originales, por lo que estas filas que dieron error en Oracle fueron ignoradas y no formaron parte de la tabla.

7. EVALUACIÓN

En este apartado se detalla la experimentación realizada para realizar la evaluación de rendimiento de ambos sistemas de bases de datos a partir de las consultas diseñadas previamente.

7.1. Diseño de la experimentación

En primer lugar se han establecido unas pautas a seguir a la hora de realizar las consultas en ambos sistemas. Estas pautas se han establecido con el fin de intentar que ambos sistemas ejecuten las consultas en las condiciones más similares posibles, consiguiendo así que la comparativa sea más real.

Algunas de estas pautas son las siguientes:

- Ejecutar las consultas en cada sistema de la manera más aislada posible. Esto quiere decir que a la hora de ejecutar las consultas, se ha dejado el procesador y la memoria lo más libres posibles con la finalidad de ejecutar las consultas de manera aislada.
- Diseñar las consultas de tal manera que se obtengan los mismos resultados en ambos sistemas. Este es obvio, ya que al trabajar con la misma base de datos, independientemente de que los datos estén estructurados o no, ante una misma consulta deberían devolver los mismos resultados.
- Adicionalmente, cuando se han ejecutado las consultas también se ha eliminado la conexión del equipo a Internet, para evitar ruido producido por procesos en segundo plano de actualizaciones o similares.
- Ambos sistemas de bases de datos se han instalado en la máquina local, evitando máquinas virtuales o simuladores, ya que este tipo de sistemas, aunque son útiles para probar pueden originar mediciones poco realistas debido a que son procesos que también se ejecutan en el procesador y por lo tanto consumen recursos.
- Se han utilizado las versiones más recientes disponibles de ambos sistemas de bases de datos buscando contar con las últimas optimizaciones y mejoras realizadas por los equipos desarrolladores de las mismas. Concretamente, de Oracle se ha utilizado la versión 12C (Release 2) y la versión 3.6 de MongoDB.
- A la hora de calcular el tiempo de ejecución de cada consulta, se ha ejecutado cada una un total de cinco veces y luego se ha hecho la media de los tiempos. Esto se ha hecho así porque con una única ejecución los tiempos podían ser poco realistas debido a datos mantenido en caché u otros procesos que estuvieran utilizando el procesador en ese momento. Para ello se ha realizado un script en JavaScript para

MongoDB que ejecuta cada consulta cinco veces y obtiene la media de los tiempos de ejecución. Asimismo, se ha hecho un script en SQL para Oracle con la misma finalidad.

- El entorno de ejecución de toda la experimentación siempre ha sido el mismo (similar para ambos sistemas).
- No se han simulado distintos nodos para el almacén NoSQL ya que en este proyecto se buscaba comparar la eficiencia de ambos sistemas ejecutándolos en un único equipo.

Por último, el equipo utilizado para la realización de la experimentación ha sido un iMac del año 2011 con un procesador i5-2400s a 2.5GHz de 4 núcles y 8GB de memoria RAM. Las pruebas se han ejecutado sobre una partición dedicada de 250GB en la que se había instalado el sistema operativo Windows 7.

7.2. Ejecución de las pruebas

En este punto se detalla la implementación de las consultas definidas en el punto 5.4 en ambos sistemas así como los resultados obtenidos para cada una de ellas. Para ello se sigue una estructura en la que se presenta, para una misma consulta, su implementación en Javascript (requisito RP-10: Implementación de las consultas en Javascript) y SQL (requisito RP-11: Implementación de las consultas en SQL) seguido de los resultados que ha arrojado cada una de ellas (requisito RP-12: Ejecución de consultas) en su entorno operacional y una breve explicación de dichos resultados.

7.2.1. Consulta 1: Número de películas producidas en la región de Francia

En esta consulta se obtiene el número de películas que se han producido en la región "FR" (Francia).

```
db.akas.aggregate([{
    $match: {
            "region": "FR"
    }
},
{
    $group: {
            _id: "$region",
            total: {$sum: 1}
    }
}
]);
```

Fig. 7.1. Implementación Consulta 1 en MongoDB

```
SELECT COUNT(tconst) as TOTAL, region
FROM OTROS_NOMBRES
WHERE region = 'FR'
GROUP BY (region);
```

Fig. 7.2. Implementación Consulta 1 en Oracle

A continuación se muestran los resultados obtenidos al ejecutar la consulta:

Fig. 7.3. Tiempos de la primera consulta - Sin índices

En primer lugar, con el índice que crea por defecto MongoDB en el atributo "_id" y con la clave primaria de Oracle creada sobre el campo tconst, observamos que este segundo tarda en torno a la mitad de tiempo pese a que en Oracle esta consulta requiere realizar una join de dos tablas grandes. Esto es debido a que las claves primarias en Oracle están optimizadas para proporcionar un acceso más rápido a los datos y poder trabajar con ellos mejor.

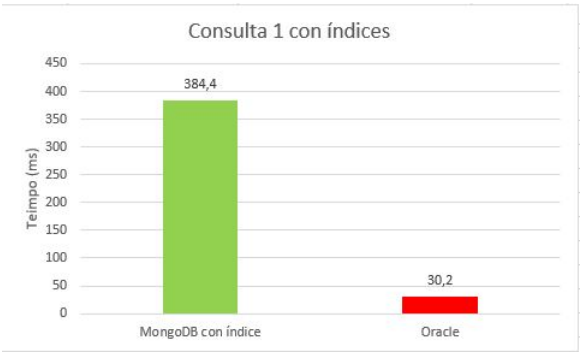


Fig. 7.4. Tiempos de la primera consulta - Con índices

En esta segunda imagen se muestra los resultados obtenidos tras crear el índice sobre el campo region tanto en MongoDB como en Oracle. En esta segunda versión de la consulta vemos que sigue siendo Oracle más rápido porque MongoDB no utiliza los índices para las operaciones de agregación, sino que únicamente los utiliza con los operadores `$match` y `$sort` [47]. Por otro lado, Oracle sí que utiliza estos índices y así consigue optimizar mejor las operaciones de agregación.

7.2.2. Consulta 2: Géneros de la película Titanic

Con esta consulta se obtiene los distintos géneros que tiene una película concreta. La película escogida ha sido el título "Titanic".

```
db.titleBasic.find(
    {'tconst':'tt0120338'},
    {'_id':0, 'endYear':0, 'runtimeMinutes':0, 'titleType':0}
);
```

Fig. 7.5. Implementación Consulta 2 en MongoDB

```
SELECT FILM.tconst, FILM.primaryTitle, FILM.originalTitle, FILM.
   isAdult, FILM.startYear, GENRES.genre
FROM FILM
INNER JOIN GENRES
ON FILM.tconst = GENRES.tconst
WHERE FILM.TCONST = 'tt0120338';
```

Fig. 7.6. Implementación Consulta 2 en Oracle

A continuación se muestran los resultados obtenidos a la hora de ejecutar la consulta:

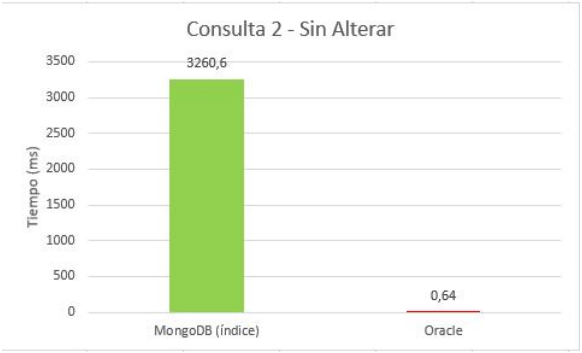


Fig. 7.7. Tiempos de la segunda consulta - Sin índices

En la primera imagen, utilizando el índice creado por defecto por MongoDB sobre el campo autogenerado "_id" de tipo ObjectId podemos observar cómo el tiempo es mucho mejor en Oracle. Esto es debido a que en Oracle, al utilizar la directiva `WHERE` se filtra por la clave primaria de la tabla "FILM" con lo que las operaciones son muy rápidas dado que por defecto la clave primaria es un índice. En cuanto a MongoDB, observamos cómo, a pesar de filtrar por el campo tconst, al no ser este un índice, la búsqueda es mucho más lenta.

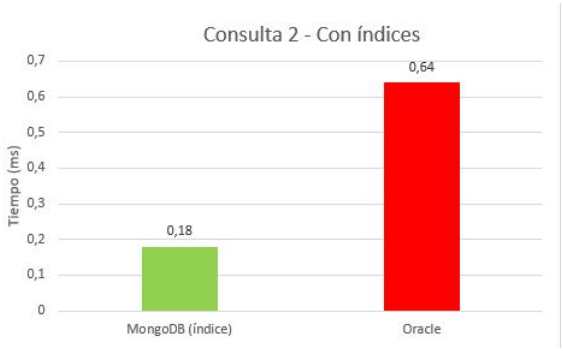


Fig. 7.8. Tiempos de la segunda consulta - Con índices

Sin embargo, al crear un índice en MongoDB sobre el campo tconst, vemos cómo el tiempo de ejecución baja notablemente. Tiene sentido que tarde menos en MongoDB ya que, dada la estructura original de los documentos, en esta consulta únicamente tendría que localizar el documento y devolver el atributo géneros, el cual es un array. Esto debería ser mucho más rápido que en Oracle, en el cual para obtener el mismo resultado debe hacer una join de dos tablas y buscar las que coincidan, lo cual es mucho más tedioso.

7.2.3. Consulta 3: Ordenación de todas las película que cumplen una serie de características

Mediante esta consulta se obtienen todas las películas que no tienen una duración inferior a 100 minutos o bien que no cumplen la condición de: haberse producido a partir del año 2000, tengan una duración inferior a 120 minutos y que estén catalogadas como para adultos. Finalmente, el resultado es ordenado en orden ascendente por el campo "startYear".

```
db.titleBasic.find(
{$nor:
    [ {runtimeMinutes: {$lt: 100}},
            {$and: [
                                    {startYear: {$gt: 2000}},
                                    {runtimeMinutes: {$lt: 120}},
                                    {isAdult: "1"}
                            ]
            }
    ]
}).sort({startYear:1})
```

Fig. 7.9. Implementación Consulta 3 en MongoDB

```
1       SELECT tconst, runtimeMinutes, startYear, isAdult
2       FROM FILM
3       WHERE NOT (runtimeMinutes<= 100 OR (startYear > 2000 AND
           runtimeMinutes < 120 AND isAdult=1))
4       ORDER BY startYear ASC
```

Fig. 7.10. Implementación Consulta 3 en Oracle

Antes de analizar los resultados de esta consulta resulta imprescindible hacer una aclaración; al ejecutar en la consola de MongoDB la consulta con el código tal y como se muestra un poco más arriba, al cabo de unos segundo se indicaba un error comunicando que no podía realizar la consulta debido a que la operación de ordenación necesitaba utilizar más memoria RAM del máximo permitido. Como posibles soluciones se indicaba o bien crear un índice o añadir un límite a la consulta.

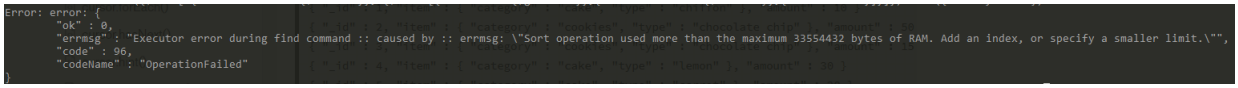


Fig. 7.11. Consulta 3 - Error de memoria

Sin embargo, en Oracle, sí que se pudo ejecutar la consulta con el mismo código mostrado anteriormente y no dio ningún tipo de error a la hora de realizar la ordenación, tardando de media un tiempo de 4914.4ms.

Fig. 7.12. Tiempos de la tercera consulta - Con límite y sin índices

A continuación, dado que MongoDB no permitió ejecutar la consulta original diseñada, se optó buscar un límite que permitiera ejecutar la consulta en ambos sistemas con el fin de que al tener que ordenar un número menor de resultados no tuviera ningún problema. A base de probar llegamos a que con un límite de 145000 registros nos permitía realizar la consulta. Como se puede observar en la imagen superior, en esta consulta la ejecución en Oracle resultaba mucho más rápida que en MongoDB.

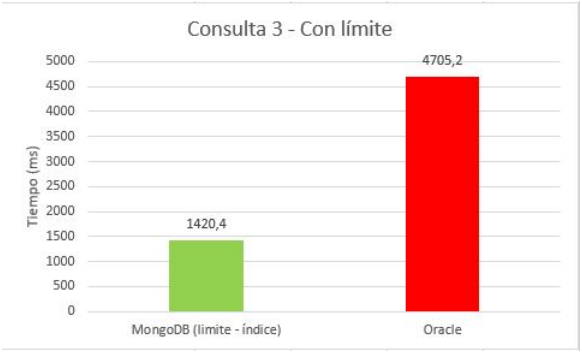


Fig. 7.13. Tiempos de la tercera consulta - Con índices y con límite

Por último, comentar que para ejecutar la consulta en igualdad de condiciones, además de incluir en ambas consultas un límite de 145000 registros, en MongoDB también se decidió crear un índice sobre el atributo tconst, que sería el equivalente a la clave primaria en Oracle. Así, con estas consideraciones, podemos observar cómo en este caso MongoDB tarda considerablemente menos que Oracle. Esto es debido al uso que hace MongoDB de la memoria y a la utilización del índice, ya que en este caso sí que utiliza el índice para acceder a los datos y estos ya se encuentran ordenados en memoria.

7.2.4. Consulta 4: Cálculo de la edad de todas las personas fallecidas entre dos años

En esta consulta se calcula la edad de todas las personas fallecidas entre el año 1918 y el año actual. A continuación se realiza una ordenación por este nuevo campo calculado.

```
db.nameBasic.aggregate([
    {$match: {
            $and: [
                    {"deathYear": {$ne: null}},
                    {"birthYear": {$gt: 1918}}
            ]
    }
    },
    {
            $project: {
                    '_id': 0,
                    'deathYear': 1,
                    'birthYear': 1,
                    'primaryName': 1,
                    age: {
                            $subtract: [
                                    "$deathYear",
                                    "$birthYear"
                            ]
                    }
            }
    },
    {
            $sort: {'age': 1}
    }
]);
```

Fig. 7.14. Implementación Consulta 4 en MongoDB

```
SELECT nconst, birthyear, deathyear, (deathyear-birthyear) as age
FROM  PERSONS
WHERE deathyear is not null and birthyear > 1918
ORDER BY age ASC;
```

Fig. 7.15. Implementación Consulta 4 en Oracle

A continuación se presentan los tiempos obtenidos para esta consulta:

Fig. 7.16. Tiempos de la tercera consulta - Sin índices

En esta primera imagen se muestran los tiempos obtenido sin realizar ninguna modificación; esto es sin crear ningún índice adicional y dejando el que implementa MongoDB por defecto. Para esta consulta vemos cómo Oracle es más rápido que MongoDB, debido seguramente a la utilización del índice de la clave primaria.

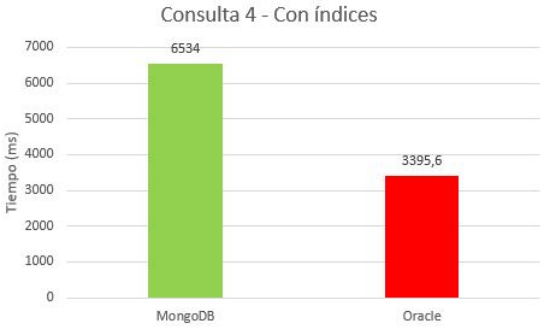


Fig. 7.17. Tiempos de la cuarta consulta - Con índices

En esta segunda imagen, se pueden observar los tiempos obtenido tras la ejecución de la consulta en cada sistema pero en esta ocasión utilizando índices. En primer lugar se decidió crear un índice en MongoDB sobre el campo “nconst”, que sería el equivalente a la clave primaria en MongoDB. Para este primer índice creado no se apreciaron mejoras en la ejecución, seguramente debido a que para esta consulta no se utiliza el campo “nconst”. En el caso de Oracle la consulta es más rápida ya que este optimiza los accesos a bloque utilizando la clave primaria.

7.2.5. Consulta 5: Agregaciones sin cálculos

En esta consulta se obtiene diferente información relativa a las películas. Para esta prueba se ha elegido la película “Titanic” y a través de varias agregaciones obtienen la nueva información. Aunque en consulta no se realice ningún cálculo con los datos, resulta especialmente interesante debido a que para conseguir la información que se busca es necesario trabajar con una cantidad importante de registros en memoria.

```
db.crew.explain("executionStats").aggregate([
    {
            $match: {
                    tconst: 'tt0120338'
            }
    },
    {
            $lookup: {
                    from: "titleBasic",
                    localField: "tconst",
                    foreignField: "tconst",
                    as: "infoPeli"
            }
    },
    {
            $unwind: "$infoPeli"
    },
    {
            $unwind: "$directors"
    },
    {
            $lookup: {
                    from: "nameBasic",
                    localField: "directors",
                    foreignField: "nconst",
                    as: "infoDirector"
            }
    },
    {
            $unwind: "$infoDirector"
    },
    {
            $project: {
                    '_id': 0,
                    'infoPeli._id': 0,
                    'infoDirector._id': 0,
            }
    }
])
```

Fig. 7.18. Implementación Consulta 5 en MongoDB

```
WITH STAGE1 AS (
SELECT PARTICIPATE.tconst, PARTICIPATE.nconst FROM
PARTICIPATE
WHERE TCONST = 'tt0120338' AND PARTICIPATE.esDirector = 1
),
STAGE2 AS (
    SELECT FILM.*, STAGE1.nconst FROM STAGE1
    INNER JOIN  FILM
    ON FILM.tconst = STAGE1.tconst
)
SELECT STAGE2.*, PERSONS.* FROM STAGE2
INNER JOIN  PERSONS
ON PERSONS.nconst = STAGE2.tconst;
```

Fig. 7.19. Implementación Consulta 5 en Oracle

A continuación se muestran los resultados obtenidos para esta consulta:

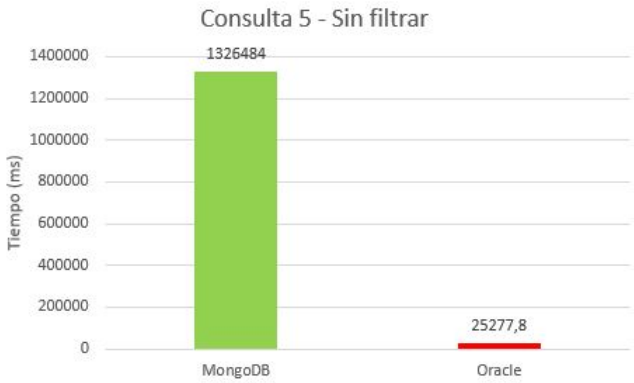


Fig. 7.20. Consulta 5 - Full Scan

En primer lugar, en esta consulta, a modo de prueba adicional, se probó a hacer una full join, es decir, no filtrar la información con where o match. Esta prueba se hizo por curiosidad, a ver cómo se comportaban ambos sistemas a la hora de trabajar en memoria con tres conjuntos de datos de gran tamaño.

Como se puede observar en la imagen, haciendo una full join, MongoDB tarda algo más de un millón de milisegundos (el equivalente a 21 minutos, aproximadamente) mientras que Oracle tarda poco más de veinticinco mil milisegundos.

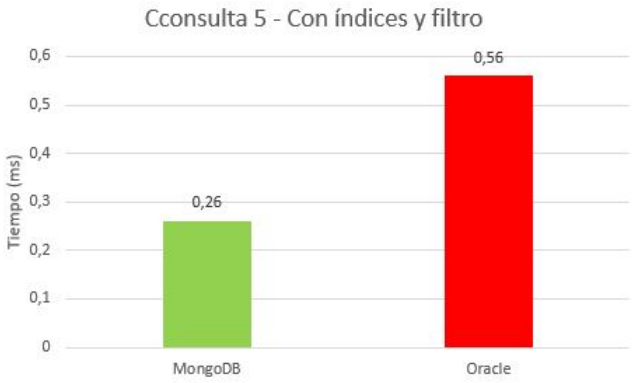


Fig. 7.21. Consulta 5 - Con índices

En este segundo caso, se creó un índice en cada campo tconst de las colecciones utilizadas en MongoDB, que sería el equivalente a las claves primarias que se habían creado en las tablas equivalentes en Oracle. En el gráfico podemos observar cómo MongoDB tarda prácticamente la mitad de tiempo que Oracle en realizar la misma consulta gracias a la creación del índice.

7.2.6. Discusión de los resultados

En este punto se hace un análisis general de los resultados obtenidos tras la ejecución de las pruebas, así como comentarios de los mismos (requisito RP-13: Análisis de los resultados).

7.2.6.1. Eficiencia e Índices

Como norma general, podemos observar que MongoDB tarda menos tiempo en casi todas las consultas que hemos realizado. No obstante, es importante destacar que la mejoría de los tiempos obtenidos en MongoDB se ha debido a la creación de los índices sobre los campos sobre los que se realizaba la consulta. Esto es debido a que gracias a la creación de los índices se consigue una ejecución más eficiente de las consultas. Si realizamos una consulta sin ningún índice, MongoDB realiza un escaneo completo de la colección buscando todas los documentos que coincidan con los criterios introducidos en la consulta. Con la creación de un índice se consigue limitar el número de documentos en los MongoDB debe buscar [48].

Los índices en MongoDB son estructuras de árboles-B que almacenan un porción más pequeña del conjunto de datos y proporcionan una forma más fácil de recorrerlos. Este índice almacena el valor de un campo o conjunto de campos ordenados por el orden del valor. Mediante esta ordenación se consigue que las consultas basadas en rangos o en condiciones sean especialmente eficientes.

No obstante, aunque mediante la creación de índices se mejoren los tiempos de las

consultas, en estas pruebas se ha creado un número pequeño de índices y sobre unas consultas muy concretas. Aunque mediante la utilización de índices mejore el tiempo de ejecución de las consultas, éstos también tienen sus puntos negativos ya que cada índice requiere un espacio mínimo en disco, el hecho de crear índices tiene un impacto negativo en el rendimiento de las consultas de escritura, ya que se deben actualizar también los índices, y por último, cuando los índices están activos, éstos consumen espacio en disco y en memoria, lo que puede tener un impacto negativo en el rendimiento de la base de datos [49].

7.2.6.2. Eficiencia y Memoria

También resulta interesante comentar el tiempo que se ha obtenido para cada una de las cinco iteraciones que se ha realizado para cada consulta tanto en MongoDB como en Oracle:

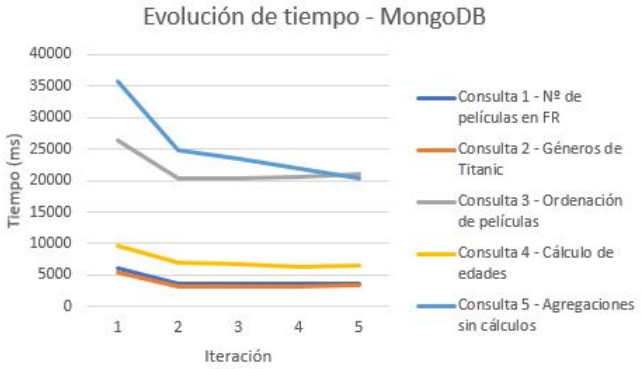


Fig. 7.22. Evolución del tiempo de las consultas en MongoDB

Como podemos observar en la imagen superior, en esta imagen se presenta la evolución del tiempo en cada ejecución de las consultas. Podemos observar que en la primera ejecución el tiempo es más alto y en las siguientes va bajando hasta establecerse más o menos constante. Esto es debido a la forma en la que MongoDB accede a los datos. MongoDB mantiene los datos más recientemente usados en memoria RAM; si se crea un índice para una consulta y todo el conjunto de datos de trabajo entra en memoria, MongoDB calcula todas las consultas en memoria, lo cual resulta mucho más rápido. Es importante comentar que MongoDB no almacena en memoria caché los resultados de una consulta, por lo tanto, aunque se repita la misma consulta varias veces siempre se calcula el resultado, pero en sucesivas ocasiones puede ser más rápido debido a que el set de datos se encuentre en memoria [50]. Por defecto, MongoDb se apropia de toda la memoria RAM libre del sistema operativo y la convierte en su memoria caché [51].

Para utilizar los datos en memoria MongoDB utiliza los denominados archivos mapeados en memoria, los cuales son archivos que el sistema operativo almacena en memoria. Así, un archivo es mapeado en memoria virtual y se establece una correlación directa byte

a byte. De esta manera MongoDB puede trabajar con los archivos como si estuvieran alojados en memoria principal, lo cual proporciona una mayor rapidez a la hora de acceder a los datos. De esta manera evita tener que acceder al disco duro para acceder a los datos, lo cual resulta mucho más lento que acceder directamente a memoria. El sistema MongoDB únicamente busca los datos en el disco duro cuando se produce un fallo de página, es decir, cuando necesita acceder a un archivo de datos y este no se encuentra en memoria [51].

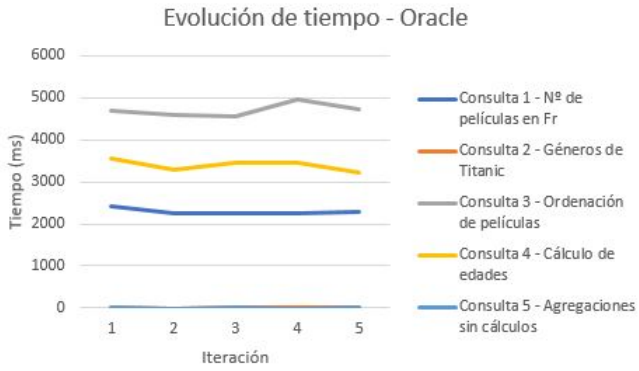


Fig. 7.23. Evolución del tiempo de las consultas en Oracle

Por otro lado, en la gráfica superior podemos observar la evolución de los tiempos en las consultas realizadas en Oracle. En este caso podemos ver cómo los tiempos se mantienen más o menos estables y no sufren grandes variaciones. Esto es debido a la utilización que hace Oracle de la memoria RAM y el disco.

Una instancia de base de datos es el conjunto de estructuras de memoria y procesos que acceden a los ficheros de datos. Dentro de las estructuras que se crean en memoria las más importantes son la PGA y la SGA [52]:

- La SGA (System Global Area o Área Global del Sistema) es un conjunto de estructuras compartidas en memoria que contienen datos e información de control de la instancia de la base de datos. Dentro de este área existen varias zonas bien diferenciadas, aunque la más importante para nosotros es la caché de buffers. En esta zona se almacenan los bloques de datos utilizados recientemente (independientemente de que se hayan confirmado o no los cambios). Mediante la utilización de este buffer se consigue reducir notablemente el número de accesos a disco, mejorando así el rendimiento.
- La PGA (Program Gloabl Area o Área Global del Programa) es una zona de memoria no compartida que contiene datos e información de control únicamente para uso del proceso servidor de Oracle. Cada vez que se arranca un proceso servidor se crea una nueva zona PGA.

No obstante, a pesar de que Oracle también dispone de gestión de memoria ésta se asigna a un espacio mucho más reducido que el que se asigna a MongoDB; a Oracle se le

asigna un buffer de memoria caché mientras que MongoDB ocupa toda la memoria RAM que tenga disponible el sistema operativo. Además, el almacenamiento de los datos en MongoDB es mucho más óptimo debido al diseño físico que tiene ya que los documentos no son más que archivos binarios almacenados de forma serial. Por otro lado, el diseño físico de Oracle está basado en bloques y estos son la unidad de comunicación entre el disco duro y la memoria; cuando se produce un fallo en memoria caché porque el dato no se encuentra en esta, se solicita un nuevo bloque al disco con el dato requerido [53]. Esto hace que estos bloques tengan que tener propiedades adicionales, para saber cómo unirlos luego, y ocupen más espacio.

Gestionar correctamente estos dos espacios de memoria es importante para mantener entre ambos un tamaño óptimo para las estructuras que se utilizan en memoria. Como norma general, Oracle recomienda establecer el modo de manejo automático de memoria, según el cual el usuario únicamente tiene que establecer unos valores de inicialización de tamaño de memoria máximo y a partir de ahí Oracle redistribuye ese espacio de memoria entre la SGA y la PGA según lo vaya necesitando [54].

7.2.6.3. Eficiencia y Diseño de las estructuras del almacén de datos

Como se ha visto, a la hora de realizar una base de datos relacional el estudio del dominio del problema para realizar un buen diseño de datos puede resultar clave para el desempeño de la base de datos. Este diseño no es proceso sencillo y conlleva realizar el modelo conceptual, el modelo relacional, la implementación y el mantenimiento de la base de datos. La principal finalidad de un diseño de base de datos es crear una serie de estructuras lógicas que permitan interpretar la información contenida en ellas y que permita sacar el máximo rendimiento. Asimismo, un mal diseño de bases de datos puede llevar a bases de datos no normalizadas que podrían dar lugar a inconsistencias en la información almacenada.

Por otro lado, en los almacenes de datos NoSQL no es necesario realizar un análisis de los datos y un diseño previo de la base de datos. Los datos almacenados en un almacén NoSQL no tienen por qué tener una estructura fija, sino que esta puede ser dinámica. De esta forma, el concepto de diseño de base de datos pierde relevancia en los entornos NoSQL debido al carácter distribuido de estos; ganando así importancia el hecho de tener un buen hardware que permita trabajar con grandes cantidades de datos.

Por poner un ejemplo relacionado con esto, en el conjunto de datos con el que hemos realizado el caso práctico, tenemos que las películas tienen un atributo llamado géneros; el cual contiene todos los géneros de una película. Este campo en la base de datos NoSQL ha sido modelado como un array en el que cada posición del mismo corresponde a un género. Por otro lado, en el sistema relacional Oracle para almacenar esta misma información se hace necesario la creación de dos tablas debido a que no cada campo debe tener un único valor. Así, se tiene que para obtener la misma información en MongoDB únicamente es necesario buscar la película por su identificador y seleccionar ese array de

géneros, lo cual es una operación prácticamente inmediata. Por otro lado, en Oracle, para obtener esta misma información se hace necesario realizar una operación de combinación de datos, subiendo a memoria una gran cantidad de registros y realizar una gran cantidad de operaciones; lo cual resulta mucho más pesado computacionalmente y por lo tanto requiere más tiempo.

Con este ejemplo se pone de manifiesto la importancia de realizar un buen diseño de la base de datos, entre otras razones porque, aunque algunas operaciones sean pesadas por naturaleza, realizar un buen diseño puede ayudar a que su tiempo de ejecución disminuya notablemente.

7.3. Resumen de conclusiones del Caso de Evaluación

Tras realizar y analizar todo el caso práctico presentado en este documento podemos sacar las siguientes conclusiones:

- Para el dataset elegido en el comienzo del caso práctico, dado que contenía eran colecciones con una estructura predefinida, ha resultado mucho más sencilla y rápida la inserción de los documentos en MongoDB ya que se pudo importar el dataset sin tener que realizar ninguna modificación. Esta inserción en Oracle ha sido mucho más lenta y tediosa ya que, dado el paradigma relacional, Oracle comprueba la integridad de la información en cada inserción.

- A la hora de realizar la inserción de los datos en MongoDB no ha sido necesario tomar ninguna consideración ni realizar ningún diseño de la base de datos que se iba a insertar. Por contra, en Oracle ha sido necesario un previo análisis de los datos de los que se disponía para realizar un diagrama conceptual para poder realizar la implementación de la base de datos con la estructura relacional necesaria.

- En la realización de las consultas, realizar consultas para MongoDB en el lenguaje Javascript llega a ser tedioso dada la diversidad de operadores que existen y la complejidad de los mismos. En contraposición, en Oracle, usando SQL, las consultas resultan mucho más sencillas e intuitivas.

- En cuanto a la estructura de los datos, podemos afirmar que es un factor mucho más importante en Oracle que en MongoDB. El rendimiento de la base de datos en Oracle puede depender en gran medida del diseño relacional realizado. Por contra, en MongoDB la estructura de los datos carece de importancia y prevalecen las características hardware del servidor de base de datos, ya que cuanta más memoria RAM tenga más documentos podrá almacenar en la misma y el rendimiento será mejor.

- A la hora de evaluar las consultas, hemos visto que según el tipo de operación, MongoDB es más rápido que Oracle. Por ejemplo, en la consulta mediante la que se obtenían todos los géneros de la película Titanic, en la colección original insertada en MongoDB esta información estaba modelada como un array por lo que el acceso era prácticamente inmediato. Sin embargo, dado el diseño relacional realizado, para obtener esta información era necesario realizar una agrupación (join) de dos tablas diferentes lo cual resulta más costoso.

- Cuando se realiza por primera vez una consulta se observa que tarda más tiempo que en las veces sucesivas. Esto es debido a que la primera vez el conjunto de datos de trabajo no se encuentra en memoria y se debe acceder a disco para obtenerlo. En sucesivas ejecuciones se ve cómo este tiempo va disminuyendo debido a que MongoDB mantiene este conjunto de datos recientemente usados en memoria. Por

otro lado, aunque Oracle también destina una parte de la memoria para el conjunto de datos recientemente usados, este espacio es mucho menor que el que utiliza MongoDB, ya que este último puede utilizar toda la memoria RAM disponible, por lo que es posible que no quepa en memoria todo el conjunto de datos y tenga que hacer un nuevo acceso a disco.

- En el caso de no estructurar la información como arrays en MongoDB sería necesario realizar, al igual que en Oracle, una combinación de varias colecciones para obtener esta información, lo que conllevaría igualmente trabajar en memoria con una gran cantidad de documentos. Mediante esta representación de la información como arrays en MongoDB se evita tener que realizar estas combinaciones de datos en memoria y se permite acceder a ellos como si fuera un atributo más del objeto, con lo que se consigue que el acceso sea mucho más rápido.

- Para este trabajo se ha optado por elegir un conjunto de datos con una estructura predefinida a pesar de que los documentos almacenados en MongoDB no tienen por qué cumplir todos este criterio. Se ha buscado un conjunto de datos con información estructurada debido a que posteriormente se iba a hacer la transformación a un sistema relacional, por lo tanto era necesario disponer de una estructura más o menos clara en los datos. En el caso de escoger un conjunto de datos con información poco estructurada, esta conversión a un modelo relacional hubiera sido mucho más compleja (incluso imposible en algunos casos) debido a que no se podrían haber modelado como entidades ni establecer relaciones entre ellas.

A la vista de estas conclusiones, podemos determinar que, a partir del dataset elegido, ha resultado más fácil trabajar con él en MongoDB debido a las características ya mencionadas. Aunque en el presente trabajo no se haya evaluado, a la hora de utilizar bases de datos de gran tamaño, MongoDB proporciona una mayor escalabilidad que la que proporciona Oracle. Esto es debido a que MongoDB está pensado para poder ser un sistema escalable y distribuido, mientras que Oracle presenta más problemas en este sentido.

También resulta importante destacar que, aunque con la creación de índices MongoDB tenía un rendimiento algo mejor a Oracle, es importante tener en cuenta que en este caso práctico los índices se han creado sobre campos en consultas muy concretas. No obstante, aunque mediante la creación de índices se consiga una cierta mejoría de rendimiento, no se debe crear una base de datos en MongoDB y hacer un índice para todos los campos de todas las consultas que se realicen, ya que los índices también tienen sus desventajas como ya se ha comentado anteriormente.

Finalmente, comentar que Oracle está muy ligado a un buen análisis y diseño de la estructura de la información que se vaya a almacenar en la base de datos, ya que el rendimiento de la misma va a depender en gran parte de ella. Por otro lado, en MongoDB no resulta tan necesario realizar un buen análisis y diseño de la estructura de la información, sino que este aspecto carece de relevancia con respecto a las características hardware de

las que se disponga. En el caso concreto de MongoDB, el rendimiento del sistema no dependerá tanto de la estructura de la información, sino de la capacidad de cálculo de la memoria RAM de la que se disponga. Por último, recordar el hecho de que Oracle está pensado para trabajar con transacciones y asegurar la consistencia de la información de la base de datos tras realizar inserciones, borrados o modificaciones así como la atomicidad de las mismas. Por otro lado, MongoDB únicamente asegura la atomicidad a nivel de documento; no a nivel de base de datos. De igual manera, la consistencia de la información en MongoDB se da de manera casual y no es algo continuado. Este aspecto también deberá ser un punto importante a tener en cuenta previamente a seleccionar un sistema de base de datos tradicional o NoSQL.

8. CONCLUSIONES Y TRABAJOS FUTUROS

8.1. Conclusiones

8.1.1. Conclusiones personales

Este trabajo de fin de grado ha supuesto un reto para mí, ya que ha supuesto realizar un trabajo relativamente largo (de unos cuatro meses de duración) de manera individual y tener que autogestionar tiempos y objetivos de acuerdo a las necesidades e inconvenientes que iban surgiendo.

De igual manera, ha resultado interesante conocer más en profundidad cómo funciona un gestor de base de datos como MongoDB u Oracle más a bajo nivel, es decir, conocer cómo funcionan y cómo gestionan estructuras en memoria o en disco.

Por otro lado, a pesar de que ya conocía previamente Oracle por alguna asignatura de la carrera, este proyecto me ha servido para aprender más sobre este sistema y profundizar en los conocimientos ya adquiridos. Asimismo, con MongoDB apenas lo conocía y tras la realización de este trabajo he aprendido a crear, trabajar y gestionar bases de datos creadas en este nuevo sistema; lo cual en un futuro puede ser útil para mi carrera.

Finalmente, comentar que para la realización de este trabajo me inscribí a dos cursos del sitio web de MongoDB [14]; uno de introducción y otro con conceptos más avanzados. Estos cursos me resultaron muy útiles e interesantes para realizar el proyecto y en un futuro me gustaría hacer alguno más y obtener las certificaciones que ofrecen.

8.1.2. Conclusiones al trabajo

El presente trabajo de fin de grado tenía como objetivo principal realizar una comparativa de rendimiento al realizar consultas en un sistema relacional tradicional y un sistema NoSQL. Por otro lado, también se pretendía presentar cómo, a partir de unas nuevas necesidades, han ido apareciendo nuevos sistemas para ayudar a cubrirlas.

Mediante la realización de este estudio se han analizado dos tecnologías aparentemente muy diferentes, como son un sistema relacional y un sistema NoSQL y se han visto las ventajas y desventajas de cada uno. Esta comparativa será un factor más a valorar a la hora de elegir entre ambas tipologías de bases de datos para el dominio concreto que se estudia y dominios de características similares.

Se ha visto cómo en Oracle realizar un análisis de la información que se quiere almacenar resulta imprescindible para identificar las necesidades que se tienen y a partir de ahí realizar un buen diseño de base de datos, ya que en los sistemas relacionales es muy

[14] https://university.mongodb.com/

importante tener un buen diseño de la estructura de los datos ya que el rendimiento de la misma va a depender en gran medida de ello.

Por otro lado, hemos visto que en el sistema NoSQL orientado a documentos MongoDB la estructura de la información almacenada no es tan importante, sino que prevalece tener un buen hardware con unas buenas prestaciones; especialmente de memoria RAM. Asimismo, proporciona mayores facilidades si se necesita almacenamiento masivo de información o una mayor escalabilidad.

No obstante, aunque ahora las bases de datos NoSQL estén más de moda y parezca que se usen más que las bases de datos tradicionales, esto no debe llevar a confusión. El sistema más adecuado será aquel que mejor se ajuste a las necesidades de cada caso particular. Como se ha mencionado a lo largo del documento, resulta esencial realizar un análisis de la información y de lo que se quiere obtener de ella. Habrá casos en los que quizás no haga falta elegir entre un sistema u otro, sino que ambos sistemas podrán utilizarse conjuntamente pero con fines distintos; el sistema tradicional se podrá utilizar para almacenar información estructurada mientras que el sistema NoSQL podría ser utilizado para, por ejemplo, realizar consultas estadísticas sobre un gran conjunto de datos.

Como última instancia, este trabajo puede servir tanto para estudiantes que quieran hacer un proyecto similar a este o que simplemente quieran informarse acerca de los sistemas NoSQL y qué aporta, como para personal de una empresa que se esté planteando montar un sistema de base de datos en su empresa y no conozca qué le puede aportar cada sistema y qué cosas le puede quitar.

8.2. Trabajos futuros

Una vez finalizado el proyecto y analizados los resultados obtenidos, así como expuestas las conclusiones, en este apartado se exponen algunas posibles líneas de trabajo para continuar o ampliar esta comparativa de rendimiento que se ha realizado:

- Se podría realizar otra comparativa de rendimiento de bases de datos utilizando otros paradigmas de almacenamiento; por ejemplo, se podría comparar el relacional con un columnar o con uno basado en grafos.
- A partir de los datos originales, hacer una transformación de los datos para almacenarlos en otro sistema, como por ejemplo Neo4j o Cassandra y hacer un estudio de los pasos necesarios para realizar la transformación de los datos.
- Con un equipo más potente al utilizado en este trabajo se podría intentar utilizar todas las colecciones que se encontraban disponibles en la página de IMDB y trabajar con el conjunto de todos los datos para realizar así un análisis más profundo.
- Utilizando los mismos paradigmas de almacenamiento que en el presente proyecto, aplicar técnicas de administración de bases de datos en ambos sistemas para ver cómo influirían dichos cambios en el rendimiento.
- Realizar una simulación de una base de datos distribuida con MongoDB; levantar varios servidores y repartir los datos entre los mismos para ejecutar las consultas y evaluar cómo varían los resultados obtenidos respecto a este trabajo.

9. WORK SUMMARY

9.1. Introduction

In recent years, due to the emergence of new technologies such as Big Data or Cloud Computing, the needs of companies have changed, since they now seek to obtain and store large volumes of information efficiently. Conversely, there have also appeared many types of applications with small needs such as concurrency in reading and writing operations with low latency or high capacity of scalability and availability. All this supposes a great challenge for the relational data bases, since, in a system of relational data bases, the most used at present, the main problem at the time of wanting to scale the system can be the relations that want to maintain between your data, stored logically in different structures.

To try to cover these new needs, a wide variety of database types have appeared. These new databases, in general, have very different characteristics from traditional relational databases, and have been called NoSQL databases (an abbreviation of Not Only SQL). Some different aspects of this new type of database are the ability to perform very fast read and write operations, capacity to store large volumes of data, easily scalable and, generally, its cost is usually much lower than that of a management system. relational databases.

All this has led to the fact that in recent years the NoSQL databases have become very popular, due in part to their ease of use and their new benefits compared to traditional management systems. The problem arises when a company needs to implement a new database and considers what type it will use, whether a traditional SQL system or a NoSQL system. Generally, this question does not have a simple answer and requires a thorough analysis. However, many companies choose one of the options because ït is fashionable.ºr because they have heard that a competitor has done so without stopping to think about their needs, in many cases incurring a serious error that can have a great impact on your business.

In this End of Degree Project, we try to capture the differences between the different types of existing databases in the market, both the traditional systems and the new NoSQL systems, which are the good and bad points of each type and carry out a comparative between two apparently opposite systems; For this, a practical case study of efficiency in complex queries implemented through a relational database and a documentary NoSQL storage system will be carried out. All this to show that many times it is not as simple as choosing one system or another randomly, but that choice requires a thorough analysis of the characteristics and needs of each business.

9.2. Objectives

The main objectives of this project are shown hereunder:

- Carrying out a study of the different types of existing databases in the market, indicating the main characteristics of each system.
- Choosing two types of storage of well-differentiated databases to perform a performance study of complex queries.
- Selection of practical case to implement and perform complex queries about the data.
- Design and implementation of a practical case and selected queries in previously selected storage systems.
- Study of the performance of the consultations implemented in both systems.

9.3. Structure of the document

This document is divided into five sections, which are summarized hereunder:

- Chapter 1 “ Introduction ”: an introduction of the document is made and the objectives that are pretended to be achieved with the realization of the project are established.
- Chapter 2 “ Project Management ”: the temporal planning is established, detailing the different stages of the project, as well as a cost analysis of the same.
- Chapter 3: “ State of the Art ”: this chapter presents the different database management systems that currently exist, both traditional and NoSQL, as well as an analysis of the socio-economic environment and the regulatory framework.
- Chapter 4 .Ánalysis": this chapter explains the entire analysis process that has been carried out to carry out the project; from the ilicitation of requirements to the analysis of the operational environment prior to the choice of the database systems that were to be used.
- Chapter 5: “ Design ”: this chapter details the design process that has been carried out during the project of both the NoSQL database and the relational database. Likewise, the queries that are going to be made in the comparison are detailed and a brief explanation of them is provided.
- Capítulo 6: “Implementación”: en este capítulo se detalla la implementación de las consultas tanto en MongoDB como en Oracle.
- Chapter 7: “ Evaluation ”: this chapter explains the design of the experimentation that has been carried out, as well as the explanation of the results obtained in the execution of the queries.
- Chapter 8 “Conclusions and future work ”: the conclusions that have been obtained from the work carried out are explained. In the same way, possible lines are established that could be used as a continuation of this project in the future.

Finally, an annex with the acronyms and terms used in the document is included; an additional annex with the installation process that has been made of MongoDB an Oracle and, finally, the GANTT diagrams that have been made in the scheduling.

9.4. Description of the work

9.4.1. Choosen software

Once the objectives of this end-of-degree project have been established, a study of the different alternatives has been made in terms of information storage paradigms and new NoSQL systems existing in the market. On the other hand, an analysis of the BASIC and ACID properties has also been carried out to evaluate the features offered by traditional systems and NoSQL systems. The CAP theorem has also been analyzed in order to assess which characteristics (Consistency, Void, Fault Tolerance) were more important for the established objectives.

Starting with the CAP theorem, it establishes that in a distributed environment with shared data, consistency, availability and partition tolerance can not be maintained simultaneously. From this, we should choose the characteristics that are most critical for the business. Given the characteristics of the present project, and given that a distributed system was not available, it has been decided to give more importance to the consistency and availability implemented by the relational models; and the consistency and fault tolerance implemented by NoSQL systems such as MongoDB or HBase. We have opted for these characteristics given that, although fault tolerance is applied in distributed environments and this feature is not relevant to the project, when choosing the consistency of the relationship systems it was necessary to choose another paradigm that would also contemplate this characteristic.

On the other hand, as expected, the relational model implements the ACID properties (Atomicity, Consistency, Isolation and Durability), beacuse an important feature of these systems is the existence of transactions. However, when choosing a NoSQL paradigm, BASE properties will be chosen for those that meet the characteristics of partition consistency and tolerance. The chosen paradigm is document-oriented,because of the relevance that this type of storage is acquiring through its flexibility and speed.

Therefore, the chosen storage paradigms to perform the comparison in this work have been relational and document-oriented paradigms. Within the model of relational data, we have worked with the Oracle database management system given its great popularity in the business world.Secondly, within the systems oriented to documents we will use the MongoDB system due to its growing popularity and use.

Finally, it is worth-mentioning that, although these two systems have been chosen for the present work, any other systems could have been chosen, as well as other storage paradigms. We have opted for the aforementioned because the study of the performance of these systems can represent more clearly the objective to be achieved : highlight the importance of choosing correctly.

9.4.2. Data used

In the first place, a data domain has been searched for comparisons. We have searched for a dataset large enough to allow complex queries and require some study of the data to work with them. After searching several websites, we chose to choose a dataset with a format that was supported by MongoDB, since it was assumed that it would be more complicated to find a dataset for this system. Finally, we chose a set of data in thematic format tsv that is available on the IMDB page.

Although the complete dataset is made up of seven collections, for this work only four of them have been chosen due to their variety of data and the possibilities of consultations that they offered. The selected collections were: Title Akas, Title Basics, Title Crew and Name Basic.

9.4.3. Design of the NoSQL database

Being a NoSQL database, the stored data does not have a fixed structure; it can be perfectly dynamic. Taking this into account, in order to create the database in MongoDB, a database design has not been carried out as such, on the one hand, due to the nature of this type of database and on the other hand because when downloading the dataset it already came with a predefined structure. This structure is shown in the following image:

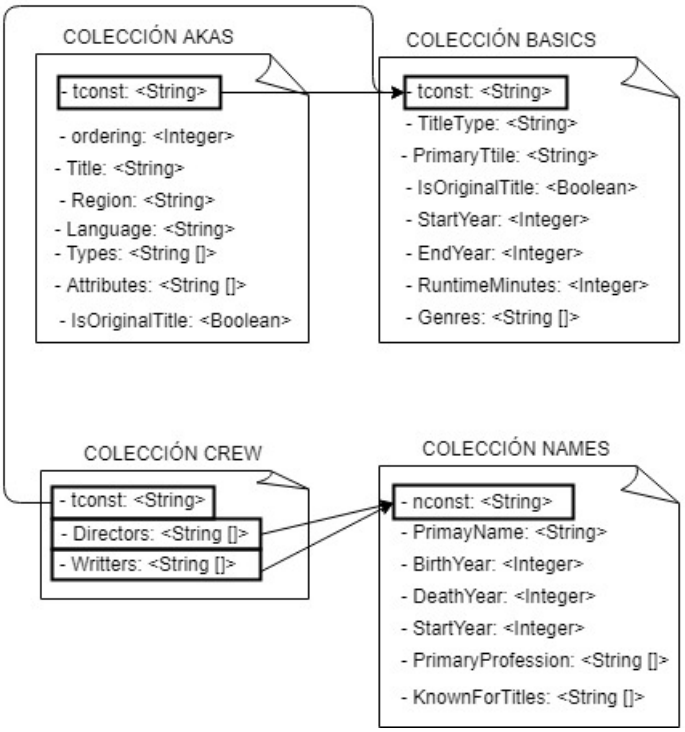


Fig. 9.1. Structure diagram in collections

As we can observe, the colections contain information related to actors, movies and directors. Although the NoSQL database does not contemplate the concept of foreign key,

we observe that all the colections repeat the field tconst, which represents an identifier for each movie. Likewise, the field nconst represents an individual identifier for each person. Also, to see the semantic assumptions, please read the point 5.3.Diseño de la base de datos relacional

9.4.4. Design of the relational database

Given that Oracle implements a relational model, the created database must follow this model. The set of download data has a prefixed structure that we have had to analyse, in order to create a conceptual design of such database in an efficient way. After this analysis, we have made the following conceptual diagram:

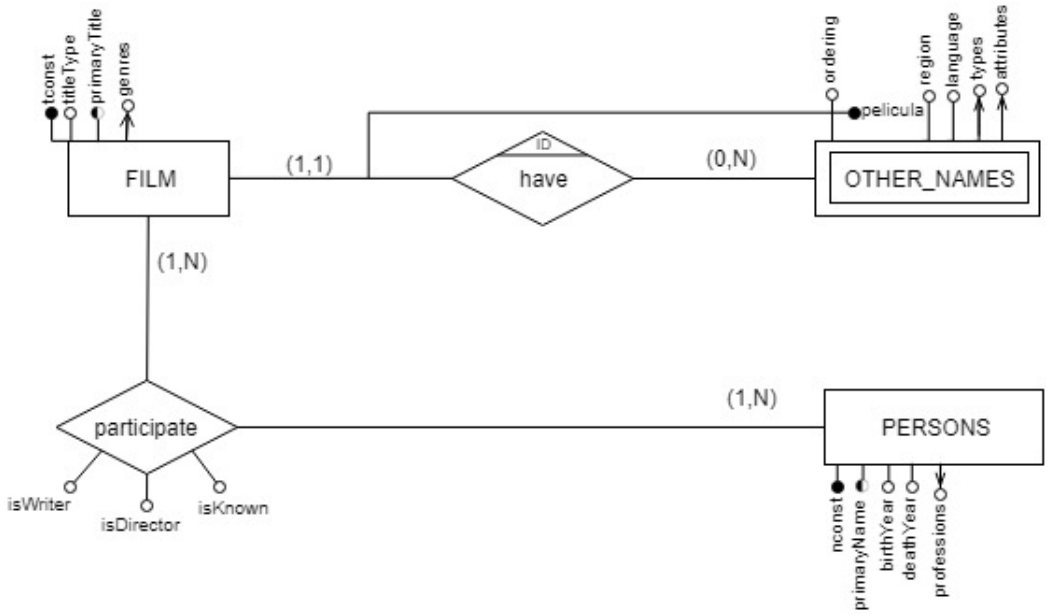


Fig. 9.2. Diagrama E/R Diagram

Although other designs could have been made, this diagram is created to find an efficient design, trying to avoid as much as possible the null values, as well as storing redundant information.

Finally, from this E / R design we have obtained the final relational diagram from which we will make the implementation in the database. This relational diagram was obtained after applying the transformation rules of E / R to relational studied in the degree. This relational diagram is shown in the following image:

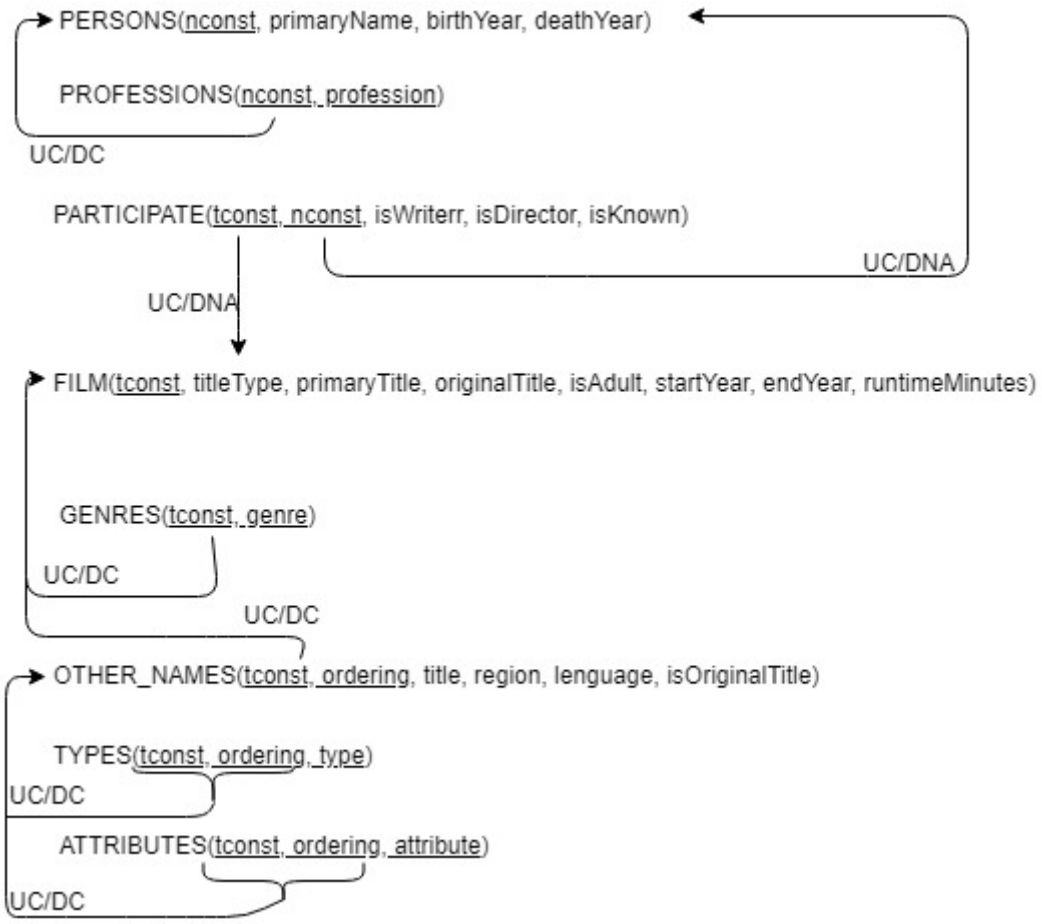


Fig. 9.3. Relational diagram

In order to maintain the consistency of the stored information, the necessary referential integrity rules have been added, even though they were not included in the original NoSQL database because they were not necessary. Also, to see the transformation of semantic assumptions, please read the point 5.3.Diseño de la base de datos relacional

9.4.5. Queries made in the comparison

In order to make this comparison, we have chosen queries that involve working with a large number of elements and that may require a large number of operations by the computer, such as data combination operations, plannings or groupings.

The chosen queries are the following:

- In the Title Akas collection, count the number of films that have been made in a certain country. It is a query that requires a grouping operation, as well as a mathematical operation.
- In the Title Basic collection, get the different genres that a movie has. This query in MongoDB requires consulting an array. On the other hand, in Oracle this query

will require a join and work in memory with a large number of records.

- In the Title Basic collection, obtain all the films that have been produced since a certain year and that fulfill a series of logical characteristics to finally order them in chronological order. This query requires the use of logical operators as well as a memory arrangement of a remarkable number of elements.
- In the Name Basic table, the ages of all the people who have been born and died between two specific years and perform an order for this new calculated value. This query requires the realization of mathematical calculations as well as an ordering in memory.
- In the Title Crew collection, from a specific film and a specific director, get more information about that film and that director. This query involves making two joins, with the Name Basic and Title Basic collections, and therefore, working in memory a large number of records.

9.4.6. Treatment and insertion of data

Before the creation of the databases and the insertion of them in each manager, it was necessary to pre-treat the data that was to be stored.

In the case of MongoDB, the main change was to treat the arrays so that they were inserted as such in the database, because, initially, it was inserted as a String separated by commas. The fields that were arrays were modified to insert the characters [] in front and behind the field and to separate each position of the array by adding double quotation marks to each member. In addition, the null characters that were initially "\\N" were modified by the word "NULL" to make it easier to work with them. To perform both tasks, two functions were created in the Javascript language that were subsequently executed from the MongoDB console.

On the other hand, in the case of Oracle, it was necessary to make a greater number of changes and more complex since the initial dataset did not have a relational structure. In this case, the null fields were changed to the reserved word "NULL" and the attributes of the original collection that were arrays were converted into N: M tables. To carry out these changes it was decided to create a script in the Python programming language, given the easiness of the program to work with text strings. Also, an SQL script was made to create the necessary structures in Oracle.

At the time of inserting the data, since in MongoDB there was no pre-set structure, the insertion was done without any problem. On the other hand, at the time of performing the Oracle insertion, there were problems due to the references between the different tables. Also, the insertion time in Oracle was much higher than that of MongoDB because when inserting the data it must check whether the reference is being used or if it does not exist and a primary key violation is taking place.

9.4.7. Implementation and execution of queries

At this point the results obtained after the execution of the tests presented above are summarized. For more information related to this point it is advisable to read the point 7.2. Ejecución de las pruebas

For the first query, when executing it in both systems, we observed that the execution time is lower in Oracle due to the existence of an index on the primary key; whereas in MongoDB this type of structure does not exist. However, since the primary key does not affect the query, but the search is done by the region field, it was decided to create an index on said field in both systems. In this case it was observed that Oracle was also faster because it optimizes aggregation operations better.

For the second query, we saw how it is initially faster in Oracle because when using the WHERE directive it is filtered by the tconst field (primary key of the table) the operations are much faster, since this is an index. As for MongoDB, we observe how searches are much slower because this is not a database. However, when creating an index on this tconst field in MongoDB we observe how time decreases considerably and is faster than in Oracle. It makes sense that it is faster in MongoDB since, in this case, it would only have to locate the specific document and return the genre attribute of it. On the other hand, in Oracle, this same query requires making a join of two tables and searching for the ones that match, which is much more complex.

In the case of the third query, it is worth mentioning that at the time of the original query in MongoDB, a memory error was obtained. This is due to the large number of documents that were asked to be ordered, while in Oracle this ordering could be done without any problem. To solve this problem and to be able to perform the query on equal terms, it was decided to add a limit on the number of documents to be returned. This limit was set at 145000 items. Once this limit was added, it was possible to observe how Oracle was still faster. However, when creating an index in MongoDB on the tconst field, which would be the equivalent of the primary key in Oracle, it was observed how the time in MongoDB decreased considerably. This was due to MongoDB's use of memory and index utilization.

For the fourth query, firstly, without creating indexes in MongoDB, we observed how Oracle is faster, probably due to the use of the index of the primary key. However, when creating the MongoDB index on the nconst field, which would be the equivalent of Oracle's primary key, we see how the time practically does not change and Oracle is still faster. This is probably due to the fact that, on the one hand, this nconst field does not intervene in the query and on the other hand, Oracle optimizes the block accesses using the primary key.

Finally, for the fifth query, we could see how to perform the query without any filter, the time MongoDB needs to make this query is triggered, because it has to keep a large amount of data in memory. On the other hand, Oracle to perform the same query takes

much less because it works with blocks. Finally, a filter was added and an index was created for each tconst field used in MongoDB, observing how time dropped considerably and became faster than Oracle. This was because, when working in memory with information, access to it is also faster.

9.4.8. Discussion of the results

As a result of the obtained results, we can observe that, in general, MongoDB takes less time in the performed consultations. However, it is important to underline that this improvement in times is due to the creation of indexes on the fields on which the consultations were carried out, since these indices achieve a more efficient execution of the queries. With the creation of an index it is possible to limit the number of documents in which MongoDB must search.

However, although the creation of indexes improves query times, these also have their negative points, since each index requires a minimum disk space. On the other hand, when indices are active, they consume disk space and memory, which can also have a negative impact on overall system performance.

It is also worth mentioning that when a query is made in MongoDB several consecutive times, the time in each execution decreases. This is due to the way in which MongoDB accesses the data since it keeps the most recently used in RAM. Also, if an index is created for a query and the entire work data set goes into memory, MongoDB calculates all queries in memory, which also makes them much faster.

On the other hand, the evolution of time after each query in Oracle remains more or less stable due to the use made by the memory and the disk. An instance of Oracle primarily creates two memory structures: the SGA and the PGA, which can be configured to exchange memory as needed. The main difference between memory management done by Oracle and MongoDB is that Oracle has a much smaller memory space, while MongoDB can use all the available RAM. Likewise, the physical design with which Oracle works is much more complex than that of MongoDB since Oracle's is based on blocks and so communication is made between the hard disk and memory. On the other hand, in MongoDB the documents are nothing more than binary files stored in a serial way, which makes it easier to work with them.

9.5. Conclusions and future works

9.5.1. Conclusions

The main objective of the present thesis was to carry out a comparative performance between a traditional relational system and a NoSQL system. On the other hand, it was also pretended to introduce how, from new needs, new systems have appeared to help cover them.

By the performance this study, two apparently very different technologies have been analyzed, such as a relational system and a NoSQL system, and the advantages and disadvantages of each have been analysed. We have observed how, in order to perform an analysis of the information that you want to store in Oracle, is essential to identify the needs you have and, from there, make a good database design, since in relational systems it is very important to have a good design of the structure of the data; the performance of it will depend to a large extent on it.

On the other hand, we have seen that in the NoSQL system oriented to MongoDB documents the structure of the stored information is not so important, but rather that good hardware with good performance prevails; especially RAM. It also provides greater facilities if massive information storage or greater scalability is needed.

However, although now NoSQL databases are more fashionable and appear to be used more than traditional databases, this should not lead to confusion. The most suitable system will be the one that best fits the needs of each particular case. As mentioned throughout the document, it is essential to analyze the information and what you want to obtain from it. There will be cases in which it may not be necessary to choose between one system or another, but both systems may be used together but with different purposes; The traditional system can be used to store structured information while the NoSQL system could be used to, for example, perform statistical queries on a large data set.

As a last resort, this work can serve both to students who want to do a project similar to this or to those who simply want to learn about NoSQL systems and what it provides, as well as for staff of a company that is considering setting up a database system in a company and do not know what each system can provide.

9.6. Future works

Once the project has been completed and the results obtained have been analyzed, as well as the conclusions presented, in this section some possible lines of work are exposed to continue or extend this performance comparison that has been made:

- Another comparison of database performance could be made using other storage paradigms; for example, you could compare the relational with a columnar or with a graph-based one.
- From the original data, make a transformation of the data to store them in another system, such as Neo4j or Cassandra and make a study of the steps necessary to perform the transformation of the data.
- With a more powerful team to the one used in this work, you could try to use all the collections that were available on the IMDB page and work with the set of all the data to carry out a deeper analysis.
- Using the same storage paradigms as in the present project, apply database management techniques in both systems to see how these changes would influence performance.
- Perform a simulation of a distributed database with MongoDB, that is, raise several servers and distribute the data among them to evaluate how the results obtained with respect to this work vary.

BIBLIOGRAFÍA

[1] I. Sommerville, *Ingeniería de Software*, ninth. Pearson, 2011.

[2] U. de Alicante. (mar. de 2018). Organización Física de las Bases de Datos, [En línea]. Disponible en: https://rua.ua.es/dspace/bitstream/10045/3444/1/T7OF.pdf.

[3] M. Marqués. (jun. de 2015). Sistemas de ficheros, [En línea]. Disponible en: http://www3.uji.es/~mmarques/f47/teoria/tema1.pdf.

[4] L. y Ciencias de la Computación - Universidad de Málaga. (). Tipos de bases de datos, [En línea]. Disponible en: http://www.lcc.uma.es/~galvez/ftp/bdst/Tema2.pdf.

[5] A. I. University. (mar. de 2017). Modelos de datos, [En línea]. Disponible en: https://www.aiu.edu/cursos/base%5C%20de%5C%20datos/pdf%5C%20leccion%5C%202/lecci%5C%C3%5C%B3n%5C%202.pdf (Acceso: 02-2018).

[6] M. Marqués. (abr. de 2002). Bases de datos orientadas a objetos, [En línea]. Disponible en: http://www3.uji.es/~mmarques/e16/teoria/cap2.pdf.

[7] J. P. Torres. (abr. de 2018). Bases de datos orientadas a objetos, [En línea]. Disponible en: https://iessanvicente.com/colaboraciones/bd00.pdf.

[8] R. Elmasri y S. B. Navathe, *Fundamentos de Sistemas de Bases de Datos*, fifth. Pearson - Addison Wesley, 2007.

[9] G. de Bases de Datos Avanzadas - UC3M. (), [En línea]. Disponible en: http://ocw.uc3m.es/ingenieria-informatica/fundamentos-de-bases-de-datos/material-de-clase-1/TemaVII0CW.pdf.

[10] V. T. I. Miralles. (). Bases de Datos Distribuidas, [En línea]. Disponible en: https://iessanvicente.com/colaboraciones/BBDDdistribuidas.pdf.

[11] I. M. B. Carrera. (). Bases de datos avanzadas, [En línea]. Disponible en: http://www.webdelprofesor.ula.ve/ingenieria/ibc/bda/s21bddp.pdf.

[12] U. de Granada. (). Introducción a los Sistemas Paralelos y a la Programación Paralela, [En línea]. Disponible en: https://lsi.ugr.es/jmantas/ppr/teoria/descargas/PPR_Tema1_Introduccion.pdf.

[13] E. G. Soto. (sep. de 2001). Data Warehouse. Antecedentes, situación actual y tendencias, [En línea]. Disponible en: https://iude.webs.ull.es/investigacion/publicaciones/pdf_docs_trabajo/SERIE%20ESTUDIOS%200144.pdf.

[14] (Sep. de 2016). Data Warehouse. Antecedentes, situación actual y tendencias, [En línea]. Disponible en: http://www.evaluandosoftware.com/tratamiento-los-datos-oltp-olap-data-warehouse/.

[15] G. H. P. G. A. G. R. D. C. Giraudi. (). Aproximaciones en el estudio de Bases de Datos Espacio-Temporales y Ruteo sobre redes móviles, [En línea]. Disponible en: http://sedici.unlp.edu.ar/bitstream/handle/10915/20376/Documento_completo.pdf?sequence=1.

[16] M. A. A. D. B. E. O. Gagliardi. (2017). Bases de Datos Espacio-Temporales aplicadas en la gestión de emergencias, [En línea]. Disponible en: http://ria.utn.edu.ar/bitstream/handle/123456789/2299/GIBD2017_CONAIISI.pdf?sequence=1%5C&isAllowed=y.

[17] M. J. Antiñanco. (dic. de 2013). Bases de Datos NoSQL: Escalabilidad y alta disponibilidad a través de patrones de diseño, [En línea]. Disponible en: http://sedici.unlp.edu.ar/bitstream/handle/10915/36338/Documento_completo.pdf?sequence=5.

[18] H. G. del Busto e I. O. Y. Enríquez. (oct. de 2012). Bases de datos NoSQL, [En línea]. Disponible en: http://studylib.es/doc/8277481/bases-de-datos-nosql---revista-telem%5C%40tica.

[19] Acens. (feb. de 2014). Bases de datos NoSQL. Qué son y tipos que nos podemos encontrar, [En línea]. Disponible en: https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf.

[20] R. G. Cebreiros. (jul. de 2016). Análisis de bases de datos y tendencias tecnológicas, [En línea]. Disponible en: https://administracionelectronica.gob.es/pae_Home/pae_Biblioteca/pae_Tecnimap/pae_TECNIMAP_2004_-_Murcia/pae_COM_2004-Sistema_de_informacion_comun.html#.WyasBaczYdV.

[21] L. H. D. L. ; M. M. H. Montenegro. (). Sistemas de Base de datos Orientadas a Columnas, [En línea]. Disponible en: https://abd-ucv-computacion.wikispaces.com/Sistemas+de+Base+de+datos+Orientadas+a+Columnas.

[22] J. Han, H. E, G. Le y J. Du, "Survey on NoSQL database", en *2011 6th International Conference on Pervasive Computing and Applications*, oct. de 2011, pp. 363-366. DOI: 10.1109/ICPCA.2011.6106531.

[23] J. G. M. (abr. de 2018). Introducción a la teoría de grafos, [En línea]. Disponible en: http://www.ugr.es/~jesusgm/Curso%5C%202005-2006/Matematica%5C%20Discreta/Grafos.pdf.

[24] C. P. Álvarez, C. Pinilla y M. Bello, "Bases de datos orientadas a grafos", *Tecnología Investigación y Academia*, vol. 5, n.º 2, pp. 153-160, 2017. [En línea]. Disponible en: http://revistas.udistrital.edu.co/ojs/index.php/tia/article/view/8769.

[25] A. I. Fraga. (feb. de 2017). Internet de las Cosas: 8.400 millones de dispositivos conectados cuando acabe 2017, [En línea]. Disponible en: http://www.ticbeat.com/innovacion/internet-de-las-cosas-8400-millones-dispositivos-conectados-2017/.

[26] I. de Ingeniería del Conocimiento. (jun. de 2016). Las 7 V del Big data: Características más importantes, [En línea]. Disponible en: http://www.iic.uam.es/innovacion/big-data-caracteristicas-mas-importantes-7-v/.

[27] A. P. Herrera. (ene. de 2015). Predecir en un 70 % un crimen futuro utilizando 'big data', [En línea]. Disponible en: http://www.elmundo.es/economia/2015/01/21/54be9d42ca474188098b456f.html.

[28] (Nov. de 2016). Cómo 'The Walking Dead' o 'NCIS' ayudaron a que Donald Trump fuera presidente, [En línea]. Disponible en: http://ecodiario.eleconomista.es/noticias/noticias/7983409/11/16/Como-The-Walking-Dead-y-NCIS-ayudaron-a-que-Donald-Trump-fuera-presidente.html.

[29] B. Cendón. (ene. de 2017). El Origen del IoT, [En línea]. Disponible en: http://www.bcendon.com/el-origen-del-iot/.

[30] D. Cantón. (oct. de 2016). DDoS de actualidad: IoT y los DNS de Dyn, [En línea]. Disponible en: https://www.certsi.es/blog/ddos-actualidad-iot-y-los-dns-dyn.

[31] (Mar. de 2011). Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal., [En línea]. Disponible en: https://www.boe.es/buscar/act.php?id=BOE-A-1999-23750.

[32] G. ÁTICO34. (feb. de 2018). ¿Qué son los datos personales especialmente protegidos?, [En línea]. Disponible en: https://protecciondatos-lopd.com/empresas/datos-especialmente-protegidos-sensibles/.

[33] J. M. Sánchez. (mayo de 2018). Las claves para entender el nuevo reglamento de protección de datos europeo, [En línea]. Disponible en: http://www.abc.es/tecnologia/redes/abci-rgpd-diez-claves-nuevo-reglamento-proteccion-datos-europeo-201805242124_noticia.html#ns_campaign=mod-lo-mas%5C&ns_mchannel=leido%5C&ns_source=tecnologia%5C&ns_linkname=%5C&ns_fee=pos-3%5C&vtm_loMas=si.

[34] M. Castro. (mar. de 2018). Esto es todo lo que sucede durante un minuto en Internet, [En línea]. Disponible en: https://www.businessinsider.es/esto-es-todo-que-sucede-durante-minuto-internet-195538.

[35] A. C., J. S. G. y M. C., "Utilidad y funcionamiento de las bases de datos NoSQL", Español, *Facultad de Ingeniería*, vol. 21, pp. 21-32, 2012. [En línea]. Disponible en: http://www.redalyc.org/articulo.oa?id=413940772003.

[36] S. G. N. Lynch. (mar. de 2002). Brewer's conjecture and the feasibility of consistent available partition-tolerant web services, [En línea]. Disponible en: https://users.ece.cmu.edu/~adrian/731-sp04/readings/GL-cap.pdf.

[37] G. G. E. Plasencia. (dic. de 2017). ToremaCAP, [En línea]. Disponible en: https://bites.futurespace.es/teorema-cap-2/.

[38] A. C. R. J. S. G. S. M. C. Cuervo, "Utilidad y funcionamiento de las bases de datos NoSQL", *Revista Facultad de Ingeniería, UPTC*, vol. 21, n.º 33, pp. 21-32, jul. de 2012.

[39] E. A. Brewer. (jul. de 2000). Towards robust distributed systems, [En línea]. Disponible en: https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf.

[40] F. de informática - Universidad Complutense de Madrid. (oct. de 2012). Transacciones y control de la concurrencia, [En línea]. Disponible en: https://www.fdi.ucm.es/profesor/fernan/DBD/apuntestema07.pdf.

[41] B. M. Sasaki. (sep. de 2015). ACID vs BASE, [En línea]. Disponible en: https://neo4j.com/blog/acid-vs-base-consistency-models-explained/.

[42] A. M. I. Maqueda, "Tema 4: El Modelo E/R", Transparencias de la asignatura Diseño y Administración de Bases de Datos.

[43] ——, "Tema 4: Transformación del Modelo E/R al Relacional", Transparencias de la asignatura Diseño y Administración de Bases de Datos.

[44] K. Chodorow, *MongoDB: The Definitive Guide*, second. O'REILLY, 2013.

[45] Mongo. (ene. de 2018). Documents, [En línea]. Disponible en: https://docs.mongodb.com/manual/core/document/.

[46] MongoDB. (ene. de 2018). ObjectId, [En línea]. Disponible en: https://docs.mongodb.com/manual/reference/method/ObjectId/.

[47] ——, (ene. de 2018). Pipeline Operators and Indexes, [En línea]. Disponible en: https://docs.mongodb.com/manual/core/aggregation-pipeline/#pipeline-operators-and-indexes.

[48] ——, (ene. de 2018). Indexes, [En línea]. Disponible en: https://docs.mongodb.com/manual/indexes/.

[49] ——, (ene. de 2018). Indexes, [En línea]. Disponible en: https://docs.mongodb.com/manual/core/data-model-operations/#data-model-indexes.

[50] ——, (ene. de 2018). Cache, [En línea]. Disponible en: https://docs.mongodb.com/manual/faq/fundamentals/#does-mongodb-handle-caching.

[51] ——, (ene. de 2018). Cache, [En línea]. Disponible en: https://docs.mongodb.com/manual/faq/storage/#wiredtiger-storage-engine.

[52] O. Inc. (). Memory Architecture, [En línea]. Disponible en: https://docs.oracle.com/database/121/CNCPT/memory.htm#CNCPT007.

[53] ——, (ene. de 2018). Data Blocks, Extents, and Segments, [En línea]. Disponible en: https://docs.oracle.com/cd/B19306_01/server.102/b14220/logical.htm.

[54] ——, (). Automatic Memory Management, [En línea]. Disponible en: https://docs.oracle.com/database/121/CNCPT/cncptdba.htm#GUID-D39DB708-CC94-4EE6-ACDA-ACED36DA4DA5.

[55] Jobtonic. (jun. de 2018). Salario: Analista de datos en España, [En línea]. Disponible en: http://espana.jobtonic.es/salary/26526/16078.html.

[56] Microsoft. (oct. de 2012). Propiedades ACID, [En línea]. Disponible en: https://msdn.microsoft.com/es-es/library/aa719484(v=vs.71).aspx.

[57] Amazon. (abr. de 2018). ¿Qué es NoSQL?, [En línea]. Disponible en: https://aws.amazon.com/es/nosql/.

[58] P. Networks. (mar. de 2017). SQL vs NoSQL, [En línea]. Disponible en: https://www.psychz.net/client/blog/es/in-comparison-sql-vs-nosql.html.

[59] Baoss. (ene. de 2015). Las 4 V's del Big Data, [En línea]. Disponible en: https://www.baoss.es/las-4-vs-del-big-data/.

[60] D. Evans. (abr. de 2011). Internet of Things. La próxima evolución de Internet lo está cambiando todo, [En línea]. Disponible en: https://www.cisco.com/c/dam/global/es_es/assets/executives/pdf/Internet_of_Things_IoT_IBSG_0411FINAL.pdf.

[61] (Feb. de 2012). ¿Qué hay que hacer para cumpllir con la LOPD?, [En línea]. Disponible en: http://www.cuidatusdatos.com/infocomocumplirlopd.html#0000009b150b50407.

[62] (Feb. de 2018). Reglamento General de Protección de Datos, [En línea]. Disponible en: https://rgpd.es/.

[63] MongoDB. (feb. de 2018). Install MongoDB Community Edition on Windows, [En línea]. Disponible en: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/.

[64] ——, (feb. de 2018). MongoDB Download Center, [En línea]. Disponible en: https://www.mongodb.com/download-center?_ga=2.103000325.442348916.1528039256-12070933.1528039255#production.

[65] Oracle. (mayo de 2018). Oracle Database Software Downloads, [En línea]. Disponible en: http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index-092322.html.

ANEXO I. GLOSARIO DE TÉRMINOS Y ACRÓNIMOS

ACID: *Atomicity, Consistency, Isolation y Durability - Atomicidad, Consistencia, Aislamiento y Durabilidad*. Acrónimo proveniente de la voz inglesa que hace referencia a las propiedades que permiten clasificar las transacciones en las bases de datos relacionales.

ARCO: Acceso, Recuperación, Cancelación y Oposición. Hace referencia al conjunto de derechos a través de los cuales la LOPD garantiza a las personas el control sobre sus datos personales.

BASE: *Basic Availability, Soft Estate - Disponibiliad Básica, Estado Suave*. Acrónimo proveniente de la voz inglesa que hace referencia a la relajación de las propiedades ACID implementadas en las bases de datos NoSQL.

BSON: *Binary Javascript Object Notation - Notación Binaria de Objetos Javascript*. Acrónimo proveniente de la voz inglesa que hace referencia a la representación codificada en binario de un documento en formato JSON.

CAP: *Consistency, Availavility and Partition Tolerance - Consistencia, Disponibilidad y Tolerancia a Fallos*. Acrónimo proveniente de la voz inglesa que hace referencia al teorema según el cual un sistema distribuido no puede garantizar simultáneamente las propiedades consistencia, disponibilidad y tolerancia a fallos.

CPU: *Central Process Unit - Unidad Central de Procesamiento*. Acrónimo proveniente de la voz inglesa que hace referencia al hardware dentro de un ordenador que es encargado de interpretar las instrucciones de un programa informático.

CSV: *Comma Separated Values - Valores Separados por Comas*. Acrónimo proveniente de la voz inglesa que hace referencia al archivo de texto que almacena los datos en forma de columnas separadas por coma. Las filas se distinguen por un salto de línea.

DNS: *Domain Name System - Sistema de Nombres de Dominio*. Acrónimo proveniente de la voz inglesa que hace referencia a la tecnología utilizada para resolver los nombres de las redes, es decir, para conocer la dirección IP de la máquina en la que se aloja un dominio.

ECTS: *European Credit Transfer System - Sistema Europeo de Transferencia y Acumulación de Créditos*. Acrónimo proveniente de la voz inglesa que representa la cantidad de trabajo que realiza el estudiante para cumplir con los objetivos del programa establecido por la universidad.

LOPD: Ley Orgánica de Protección de Datos. Ley orgánica española que tiene por objetivo garantizar y proteger los derechos de los ciudadanos sobre sus datos cuando son almacenados por empresas.

JSON: *Javascript Object Notation - Notación de Objetos Javascript*. Acrónimo prove-

niente de la voz inglesa que representa formato de texto ligero utilizado para intercambiar datos.

NoSQL: *Not Only SQL - No Sólo SQL*. Acrónimo proveniente de la voz inglesa utilizado para hacer referencia a la amplia gama de gestores de bases de datos caracterizados, entre otros, por no utilizar como lenguaje de consulta SQL.

PGA: *Program Global Area - Área Global del Programa*. Acrónimo proveniente de la voz inglesa que hace referencia a la región de memoria que contiene datos e información de control para un proceso servidor.

RAM: *Random Access Memory - Memoria de Acceso Aleatorio*. Acrónimo proveniente de la voz inglesa que hace referencia a la memoria principal de un dispositivo donde se almacenan todos los datos y la información de los programas que se encuentran en ejecución.

RGPD: Reglamento General de Protección de Datos. Normativa europea que regula el tratamiento de los datos de los ciudadanos de la Unión por parte de las empresas independientemente de su país de origen.

SGA: *System Glogal Area - Área del Sistema Global*. Acrónimo proveniente de la voz inglesa que hace referencia a un grupo de estructuras compartidas en memoria que contienen datos e información de control para una instancia de Oracle.

SGBD: Sistema Gestor de Bases de Datos. Conjunto de programas informáticos que suministran a los medios necesarios para describir y manipular los datos almacenados en una base de datos.

SQL: *Structured Query Language - Lenguaje Estrucutrado de Consultas*. Acrónimo proveniente de la voz inglesa que hace referencia al lenguaje estándar de consulta para acceder a bases de datos relacionales que permite realizar diversos tipos de operaciones en ellas.

TFG: Trabajo de Fin de Grado. Trabajo individual que debe realizar todo alumno de la Universidad Carlos III de Madrid para obtener el título de los estudios que esté cursando.

TSV: *Tab Separated Values - Valores Separados por Tabulaciones*. Acrónimo proveniente de la voz inglesa que hace referencia al formato de archivos de texto para representar datos en forma de tabla con las que las columnas se separan por tabulaciones y las filas por saltos de línea.

XML: *Extensible Markup Language - Lenguaje de Marcado Extensible*. Acrónimo proveniente de la voz inglesa que hace referencia al formato universal para datos y documentos estructurados.

YAML: *Yet Another Markup Language - Tan Sólo Otro Lenguaje de Marcado*. Acrónimo proveniente de la voz inglesa que hace referencia al formato de serialización de datos diseñado para ser leído y escrito por humanos.

ANEXO II. MANUAL DE INSTALACIÓN DE MONGODB

En este anexo se explica el proceso que se ha seguido para realizar la instalación de la base de datos MongoDB en su versión 3.6. Aunque se han seguido los pasos indicados en la propia página de MongoDB para instalar la versión Community en Windows, se puede encontrar más información en la página web [63]

- En primer lugar, descargar el archivo .msi de instalación desde la página del MongoDB Download Center: `https://www.mongodb.com/download-center?_ga=2.103000325.442348916.1528039256-12070933.1528039255#production`

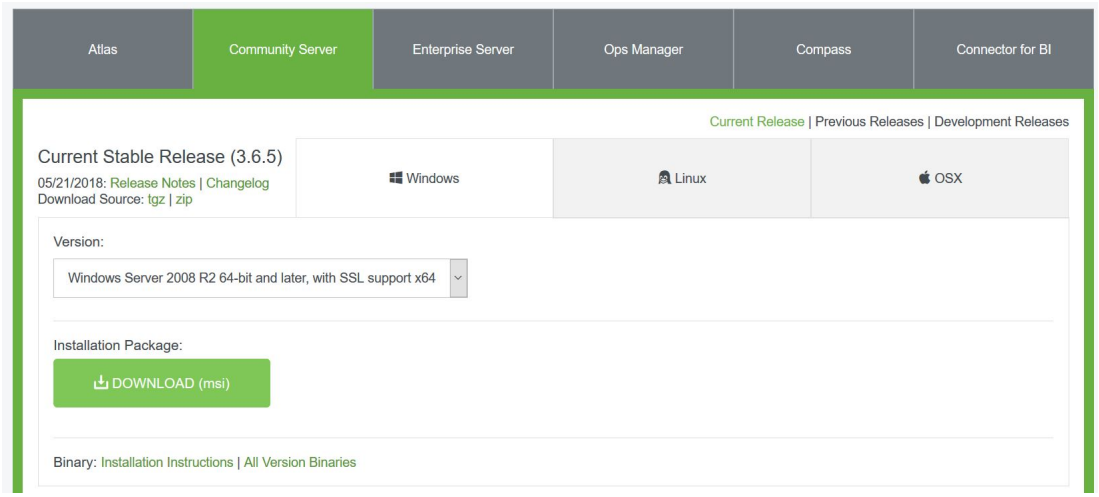


Fig. 9.4. Descarga de MongoDB [64]

- Ejecutar el archivo instalador .msi y seguir las instrucciones del asistente pulsando sobre el botón "Siguiente" en las sucesivas pantallas. Salvo que se modifique la ruta de instalación, el programa por defecto se instalará en el directorio "C:\mongodb".
- Una vez instalado MongoDB debemos añadir a la variable de entorno del sistema el directorio de instalación de mongo para que podamos lanzar los procesos desde la consola de Windows.
- Abrimos una consola pulsado la tecla Windows+R y al escribir el comando `mongod` se iniciará proceso demonio del servidor de MongoDB indicando que está esperando a nuevas conexiones en el puerto por defecto 27017.
- Por último, abrimos una nueva consola de Windows y al escribir el comando `mongo` se iniciará el proceso cliente que se conectará al servidor [15]; tras esto ya podremos comenzar a utilizar MongoDB.

[15]NOTA: es importante que estén las dos ventanas de la consola de Windows ejecutando, tanto la del cliente como la del servidor, ya que si el proceso demonio del servidor iniciado el cliente no se podrá conectar.

ANEXO III. MANUAL DE INSTALACIÓN DE ORACLE

En este anexo se explica el proceso que se ha seguido para realizar la instalación de la base de datos de Oracle en su versión 12C Release 2.

- En primer lugar, tras aceptar el acuerdo de licencia, descargar [16] el archivo .zip con el instalador desde la página del Oracle Database Software Downloads: `http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index-092322.html`. Desde esta página elegir la versión según el sistema operativo del que se disponga; en este caso Windows 7.

Fig. 9.5. Descarga de Oracle [65]

- Una vez descargado el archivo, descompirimir el archivo y ejecutar el archivo setup.exe contenido en el interior; el cual permite comenzar el proceso de instalación.

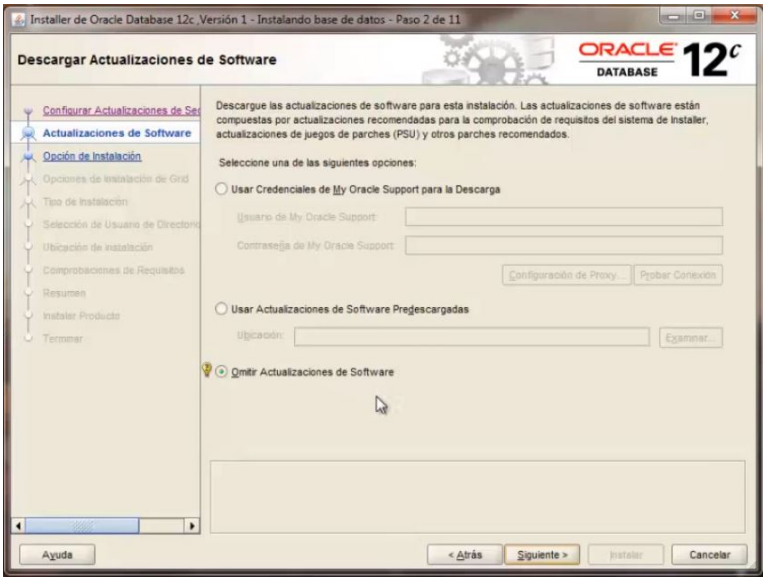


Fig. 9.6. Instalación de Oracle

[16] NOTA: para poder descargar cualquier software desde la página es necesario registrarse previamente.

- Tras ejecutarlo aparecerá una pantalla como la de la imagen superior en la cual habrá que ir completando los pasos que se presentan en el lado izquierdo y pulsando sobre el botón “Siguiente”. Tras completar todos los pasos Oracle comenzará la instalación y cuando finalice se podrá utilizar con normalidad.

ANEXO IV. DIAGRAMAS GANTT

En las páginas siguientes se muestran los diagramas de GANTT con la planificación establecida para el proyecto.

	Tarea	Duración (en días)	Fecha comienzo	Fecha finalización
1	Realización completa del trabajo fin de grado	125	22-01-2018	27-05-2018
2	Análisis	19	22-01-2018	09-02-2018
3	Diseño	23	10-02-2018	04-03-2018
4	Implementación	27	05-03-2018	31-03-2018
5	Evaluación	19	01-04-2018	20-04-2018
6	Documentación	37	21-04-2018	26-05-2018

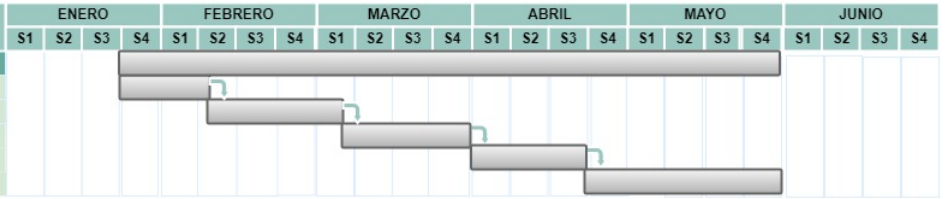


Fig. 9.7. Diagrama de Gantt - Planificación inicial del proyecto

	Tarea	Duración (en días)	Fecha comienzo	Fecha finalización
1	Realización completa del trabajo fin de grado	138	22.01.18	08.06.2018
2	Análisis	19	22-01-2018	09-02-2018
3	- Especificación de requisitos	11	22-01-2018	01-02-2018
4	- Estudio del entorno operacional	8	02-02-2018	09-02-2018
5	Diseño	26	10-02-2018	07-03-2018
6	- Búsqueda de un dataset	2	10-02-2018	11-02-2018
7	- Elección de los sistemas con los que se va a trabajar	1	12-02-2018	12-02-2018
8	- Diseño de estructura en MongoDB	2	13-02-2018	14-02-2018
8	- Análisis de la información	10	15-02-2018	24-02-2018
9	- Diseño conceptual de Oracle	5	25-02-2018	01-03-2018
11	- Estudio de operaciones a realizar	6	02-03-2018	07-03-2018
12	Implementación	33	08-03-2018	09-04-2018
13	- Tratamiento de los datos para MongoDB	5	08-03-2018	12-03-2018
14	- Creación de la base de datos en MongoDB	3	13-03-2018	15-03-2018
15	- Inserción de los datos en MongoDB	2	16-03-2018	17-03-2018
16	- Tratamiento de los datos para MongoDB	7	18-03-2018	24-03-2018
17	- Creación de la base de datos en Oracle	9	25-03-2018	02-04-2018
18	- Inserción de los datos en Oracle	7	03-04-2018	09-04-2018
19	Evaluación	22	10-04-2018	01-05-2018
20	- Diseño de la experimentación	2	10-04-2018	11-04-2018
21	- Codificación de las pruebas en MongoDB	5	12-04-2018	16-04-2018
22	- Codificación de las pruebas en Oracle	2	17-04-2018	18-04-2018
23	- Ejecución de las pruebas en MongoDB	5	19-04-2018	23-04-2018
24	- Ejecución de las pruebas en Oracle	5	24-04-2018	28-04-20188
25	- Discusión de los resultados	3	29-04-2018	01-05-2018
26	Documentación	38	02-05-2018	08-06-2018

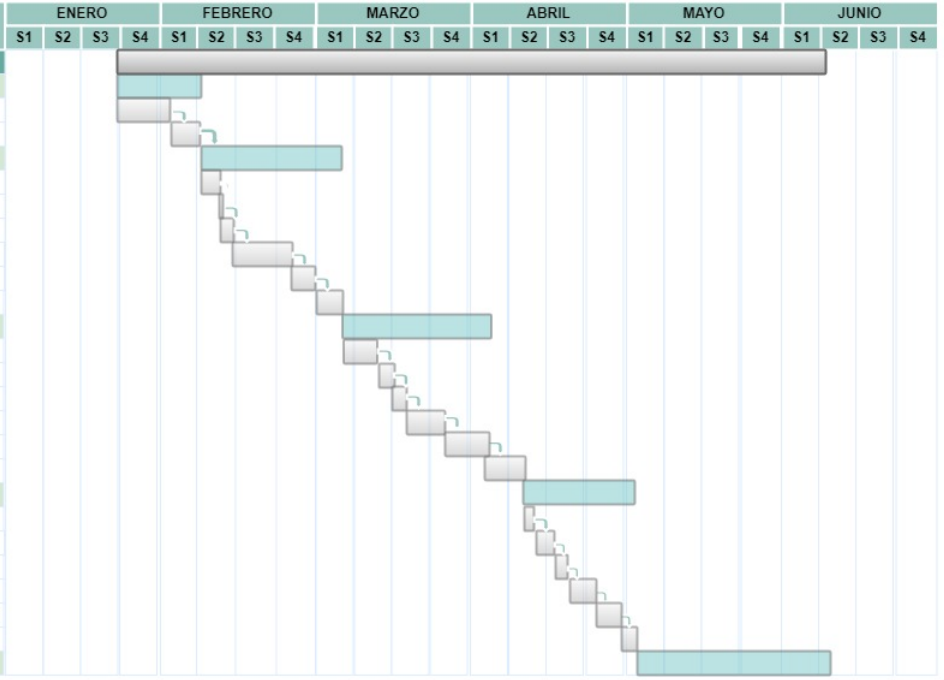


Fig. 9.8. Diagrama de Gantt - Planificación final del proyecto